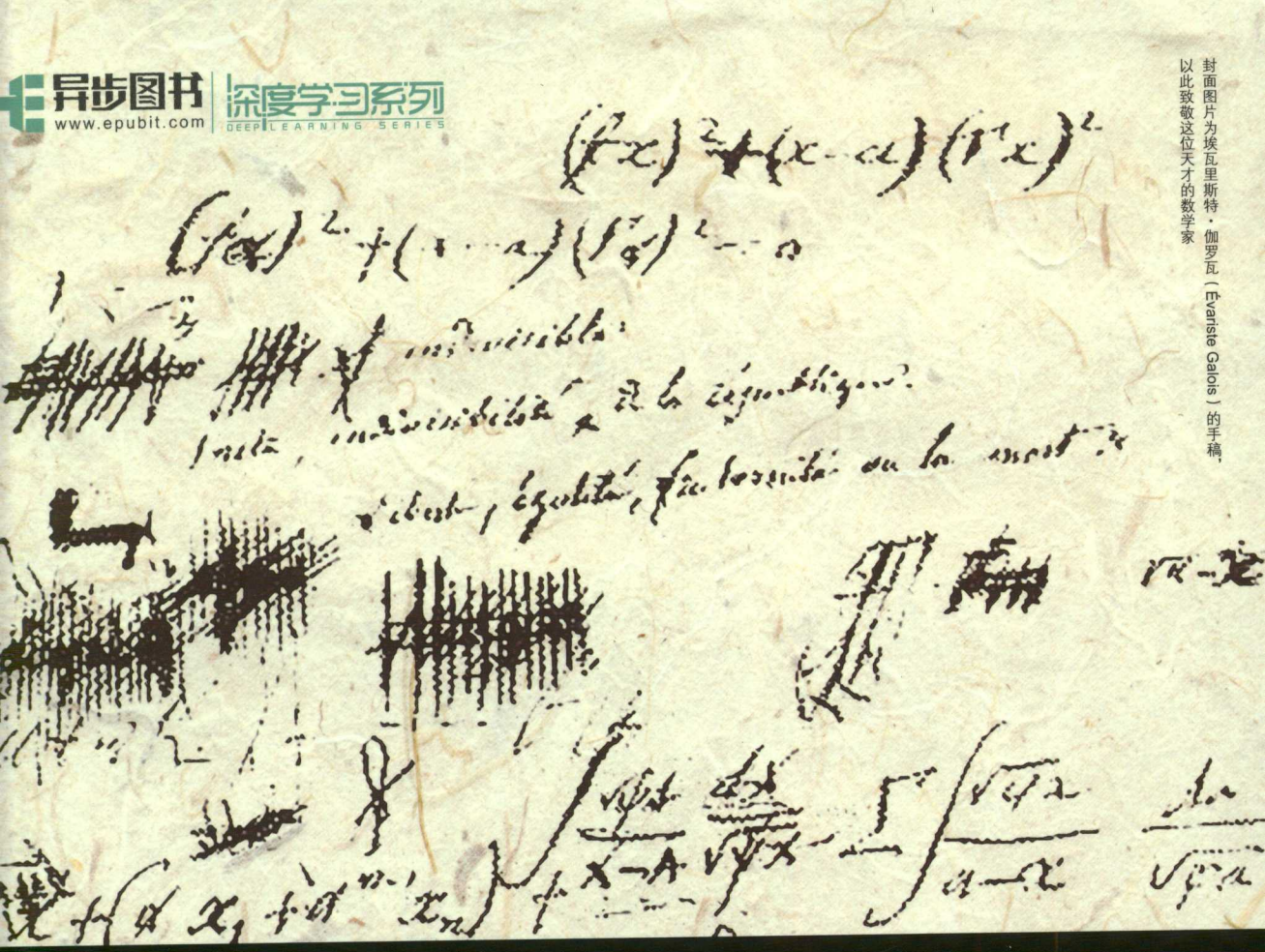
版权相关注意事项：
1、书籍版权归著者和出版社所有
2、本PDF来自于各个广泛的信息平台，经过整理而成
3、本PDF仅限用于非商业用途或者个人交流研究学习使用
4、本PDF获得者不得在互联网上以任何目的进行传播
5、如果觉得书籍内容很赞，请一定购买正版实体书，多多支持编写高质量的图书的作者和相应的出版社！当然，如果图书内容不堪入目，质量低下，你也可以选择狠狠滴撕裂本PDF
6、技术类书籍是拿来获取知识的，不是拿来收藏的，你得到了书籍不意味着你得到了知识，所以请不要得到书籍后就觉得沾沾自喜，要经常翻阅！！经常翻阅
7、请于下载PDF后24小时内研究使用并删掉本PDF

异步图书 www.epubit.com | 深度学习系列 DEEP LEARNING SERIES



封面图片为埃瓦里斯特·伽罗瓦（Évariste Galois）的手稿，以此致敬这位天才的数学家

精通数据科学：从线性回归到深度学习

唐亘◎著

★将统计学、机器学习和计算机科学融会贯通，为读者搭建系统的知识体系

★以 Python 为工具，教会读者如何建模

★详解分布式机器学习、神经网络、深度学习等大数据和人工智能的前沿方向

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

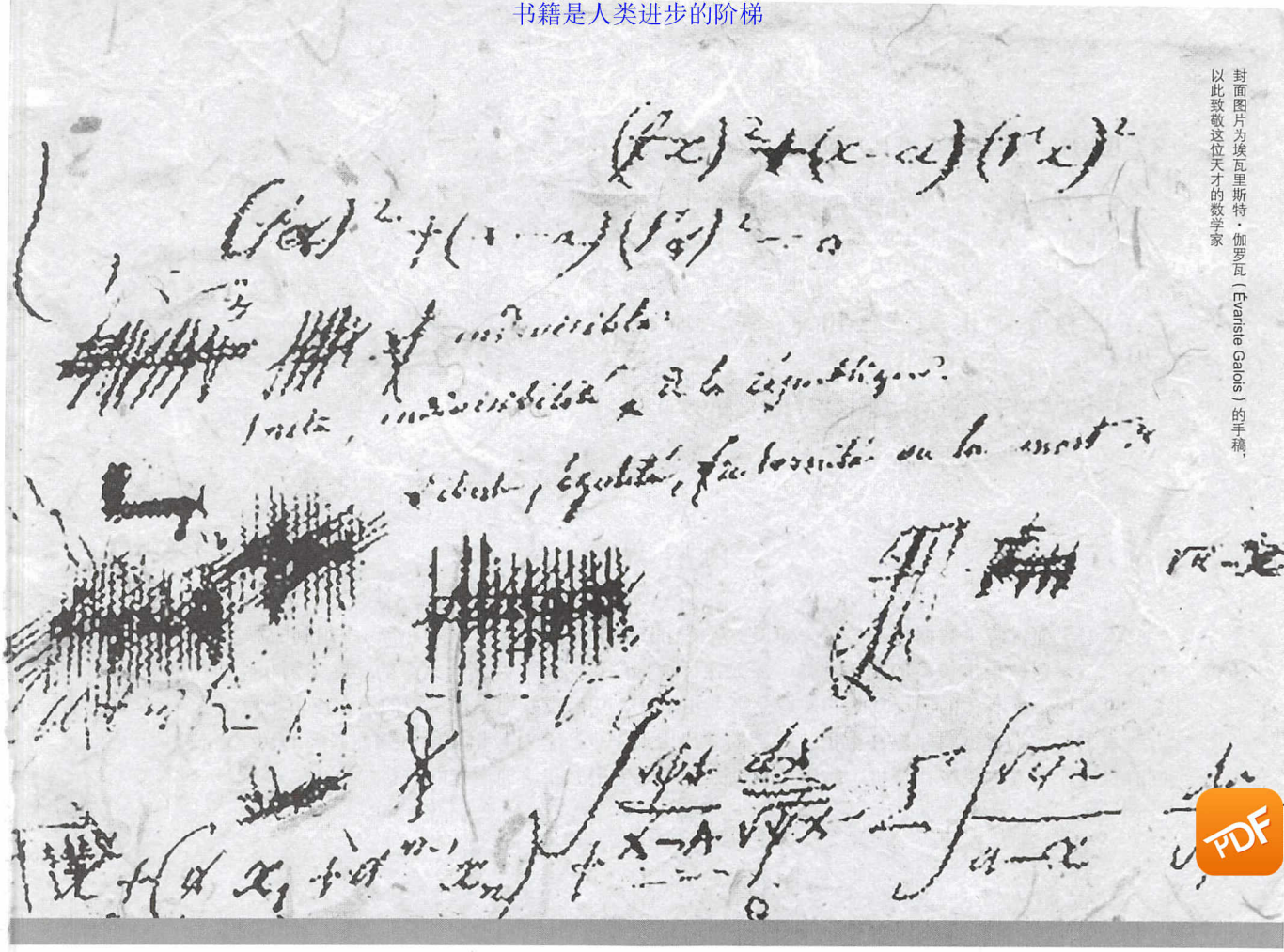
作者简介

唐亘

数据科学家，专注于机器学习和大数据，热爱并积极参与 Apache Spark、 Scikit-Learn 等开源项目。作为讲师和技术顾问，为多家机构（包括惠普、华为、复旦大学等）提供百余场技术培训。

此前的工作和研究集中于经济和量化金融，曾参与经合组织（OECD）的研究项目并发表论文，并担任英国最大在线出版社 Packt 的技术审稿人。曾获得复旦大学的数学和计算机双学士学位、巴黎综合理工大学的金融硕士学位、法国国立统计与经济管理学校的数据科学硕士学位。

书籍是人类进步的阶梯



封面图片为埃瓦里斯特·伽罗瓦（Évariste Galois）的手稿，以此致敬这位天才的数学家

精通数据科学：从线性回归到深度学习

唐亘◎著

人民邮电出版社
北京

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

图书在版编目（CIP）数据

精通数据科学 ：从线性回归到深度学习 / 唐亘著
. -- 北京 ：人民邮电出版社，2018.6
ISBN 978-7-115-47910-5

Ⅰ. ①精… Ⅱ. ①唐… Ⅲ. ①数据管理 Ⅳ.
①TP274

中国版本图书馆CIP数据核字(2018)第030929号

内 容 提 要

本书全面讲解了数据科学的相关知识，从统计分析学到机器学习、深度学习中用到的算法及模型，借鉴经济学视角给出模型的相关解释，深入探讨模型的可用性，并结合大量的实际案例和代码帮助读者学以致用，将具体的应用场景和现有的模型相结合，从而更好地发现模型的潜在应用场景。

本书可作为数据科学家和数据工程师的学习用书，也适合对数据科学有强烈兴趣的初学者使用，同时也可作为高等院校计算机、数学及相关专业的师生用书和培训学校的教材。

◆ 著　　唐　亘
责任编辑　张　爽
责任印制　马振武
◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京鑫正大印刷有限公司印刷
◆ 开本：800×1000　1/16
印张：27
字数：549 千字　　2018 年 6 月第 1 版
印数：1－4 000 册　　2018 年 6 月北京第 1 次印刷

定价：99.00 元

读者服务热线：(010)81055410　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京东工商广登字 20170147 号

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

序一

我与本书作者素不相识，读完作者发来的电子书稿后，感受到了以往在读技术类书籍时从未有过的惊喜。国内已有不少介绍大数据和机器学习的教科书和参考书，但这本书与众不同，它的重点不是传统教科书式的概念导入和各种机器学习算法的罗列，而是强调统计学、机器学习和计算机科学 3 门学科的融会贯通，试图呈现给读者关于数据科学较全面的知识体系。特别是对常用的统计和机器学习软件的详细说明，对提高在校大学生、研究生的动手能力和企业科技人员解决实际问题的能力大有裨益。

中国工程院院士，第三世界科学院院士，曾任中国科学院计算技术研究所所长

李国杰

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

序二

回首 30 年的新兴产业的来路，我们看到的或许是遵循着摩尔定律飞速增长的集成电路，或许是从互联网到移动互联网，再到物联网的更广泛的互联互通。但其背后，数据作为新兴产业的血液，其价值得到了广泛的认知和关注。早在 2011 年，我们完成了 4 篇大数据行业的前瞻报告，撰写《大数据时代的历史机遇》分析了大数据时代的产业机会与变革。后来又同申万宏源的计算机首席分析师刘洋一起勾勒了大数据产业的版图和发展路径。

如今作为一个大数据产业的实践者，我们看到大数据产业正如我们所预期的那样，成为了人工智能、虚拟现实以及区块链等新一轮新兴产业浪潮的核心，成为了传统产业转型升级的必备资源，成为了企业保持领先抑或实现弯道超车的必争之地。然而数据资源怎么用，数据模型如何建立，算法模型如何运用，成为了学界、产业界、资本界都在关注的关键问题。

本书站在数据学科的角度，融合了数学、计算机科学、计量经济学的精髓。不仅从“道”的层面为读者阐释了数据科学所要解决的核心问题——数据模型、算法模型的理论内涵和适用范围，而且从“术”的层面以常用的 IT 工具——Python 为基础，教会读者如何建模以及通过算法实现数据模型，具有很强的实操性。在此基础之上，本书还为读者详解了分布式机器学习、神经网络、深度学习等大数据和人工智能的前沿技术。相信本书将成为数据科学工作者、数据工程师、数据产业实践者的必备手册，以及想要了解和学习数据科学的人员的首选教材。

易选股金融智能证券董事长，键桥通讯董事

易欢欢

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

前　言

和武侠世界里有少林和武当两大门派一样，数据科学领域也有两个不同的学派：以统计分析为基础的统计学派，以及以机器学习为基础的人工智能派。虽然这两个学派的目的都是从数据中挖掘价值，但彼此“都不服气”。注重模型预测效果的人工智能派认为统计学派“固步自封”，研究和使用的模型都只是一些线性模型，太过简单，根本无法处理复杂的现实数据。而注重假设和模型解释的统计学派则认为人工智能派搭建的模型缺乏理论依据、无法解释，很难帮助我们通过模型去理解数据。

从历史上来看，一门学科出现相互对立的学派通常意味着这门学科处于爆发的前夜，比如 20 世纪初的经济学，凯恩斯学派和新古典经济学派的长期论战极大地促进了宏观经济学的发展，并深刻地影响了各国政府的经济政策，并由此改变了人们的生活方式。现在数据科学也处在这样相似的位置和时间节点，它已经开始并将继续改变我们的世界。

抛开这些学术上的纷争，在实际工作中应该采用哪个学派的方法来解决数据挖掘的问题呢？答案是两者都需要，而且两者都重要。在某些应用场景中，比如图像识别领域，人工智能模型有非常惊艳的表现。虽然人们还没弄清楚这些模型的工作原理，但并不妨碍它们在现实中发挥作用。事实上，人类在很多其他领域里也是这种实践先行的状态。

但在更多的应用场景中，统计学派的方法则显得更为重要。我曾在欧洲的一家保险公司里参与过一个车险定价的项目，在这个项目里，数据科学家们主要尝试了两类模型，一类是很容易解释的逻辑回归和决策树模型，另一类是较为复杂的随机森林模型。随机森林模型的预测效果更好，如果将其投入生产中，仅在法国每年就能产生数千万欧元的利润。但问题是随机森林模型难以解释，监管部门根本不接受，所以只能退而求其次，使用效果较差但更易解释的决策树模型。抛开监管层的要求不说，模型的可解释性也是非常重要的。试想一下，顾客去保险公司购买车险时，被告知需要比别人花更多的钱，而对方提供的理由是，有一个不好解释的模型预测出顾客需要付更多钱，我想大部分顾客会难以接受这样的理由和做法吧。

上述的两种建模方式虽然在处理数据的方法上有很大差异，但它们有一个共同的“物质基础”——计算机。只有借助计算机强大的运算能力，我们才能在工程上实现搭建好的模型，使之发挥作用。因此，数据科学是统计学、机器学习以及计算机科学 3 门学科的交叉，涉及的知识点和技能点很庞大且复杂。如果能将这 3 门学科融会贯通，那么就能描绘出有关数据科学的全景图，进而搭建起一个完整的知识体系，而这正是我编写本书的初衷。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

本书内容

本书按照结构共分为 13 章，主要内容如下。

第 1~3 章主要介绍数据科学要解决的问题、常用的 IT 工具 Python 以及数据科学所涉及的数学基础。

第 4~7 章主要讨论数据模型，包含 3 部分内容：一是统计中经典的线性回归和逻辑回归模型；二是计算机估算模型参数的随机梯度下降法，这是模型工程实现的基础；三是来自计量经济学的启示，主要涉及特征提取的方法以及模型的稳定性。

第 8~10 章主要讨论算法模型，也就是机器学习领域比较经典的模型，各章依次讨论了监督式学习、生成式模型以及非监督式学习。

目前数据科学的两个前沿领域分别是大数据和人工智能。

第 11 章介绍大数据中很重要的分布式机器学习。

第 12~13 章讨论人工智能领域的神经网络和深度学习。

本书除基础知识外，按照主题亦可分为 3 部分。

第 1 部分主要讨论统计学派的模型和对数据的处理方法，涉及第 4、5、7 章。

第 2 部分主要讨论人工智能学派的方法，涉及第 8、9、10、12、13 章。

第 3 部分主要介绍数据科学的工程实现，涉及第 6、11 章。

在每一章的讨论中，一般会通过一个简单的例子引出模型，然后讲解模型的理论基础，接着展示模型实现的核心代码，最后讨论模型的优缺点以及与其他模型的比较。这样既能很直观地展示模型，也能结合实际代码较深入地讨论它的细节，从而帮助读者更好地掌握和使用模型。

配套代码

针对本书，最好的阅读方法是对照每一章的示例代码，动手实现所讨论的模型。这样会极大加深读者对模型的理解，提高实践能力，否则就会像读小说一样，阅读时感觉不错，但实际使用时就无从下手了。

本书配套代码的下载地址为 https://github.com/GenTang/intro_ds，供读者借鉴和使用。

需要注意的是，为了在正文中节省篇幅，突出重点，本书所展示的代码是基于 Linux 系统下的 Python 2.7，而提供下载的配套代码则是兼容 Python 3 和 Windows 系统的。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

说明

本书中部分插图中含有未翻译的英文专有名词，原因如下。

一方面，目前相关的参考文献中没有明确且权威的中文名称与之对应，如强行翻译，难保准确，且易给读者造成误解。另一方面，对于数据科学这门学科，英文名词可能是大家更加熟悉的称呼，翻译为中文后也许会使读者在理解上更加困难。

在此说明，望各位读者理解和支持。

读者反馈

由于作者知识水平有限，书中难免存在纰漏之处，敬请各位读者朋友批评指正。请发送邮件到作者的电子邮箱 tgbaggio@hotmail.com 或本书编辑的电子邮箱 zhangshuang@ptpress.com.cn。

致谢

感谢潘健辉博士，他从行文风格和数学细节上为我提出了很多宝贵的意见。感谢我的太太安恺业女士以及我的父母，他们在本书的编写期间给了我很多鼓励。感谢李国杰院士、林晓东教授、杨卫东教授、张溪梦（Simon Zhang）先生、易欢欢先生、贾真先生、张益军先生、彭耀先生、谢佳女士以及赵甘晶女士为本书提供的帮助。感谢我的初中数学老师吴献女士对我的谆谆教诲。感谢本书的编辑张爽女士为本书的顺利出版所做的工作。需要感谢的人还有很多，限于篇幅，这里就不一一列举了。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

资源与支持

本书为异步社区出品的图书，在社区（https://www.epubit.com/）上为您提供以下资源与服务。

配套资源

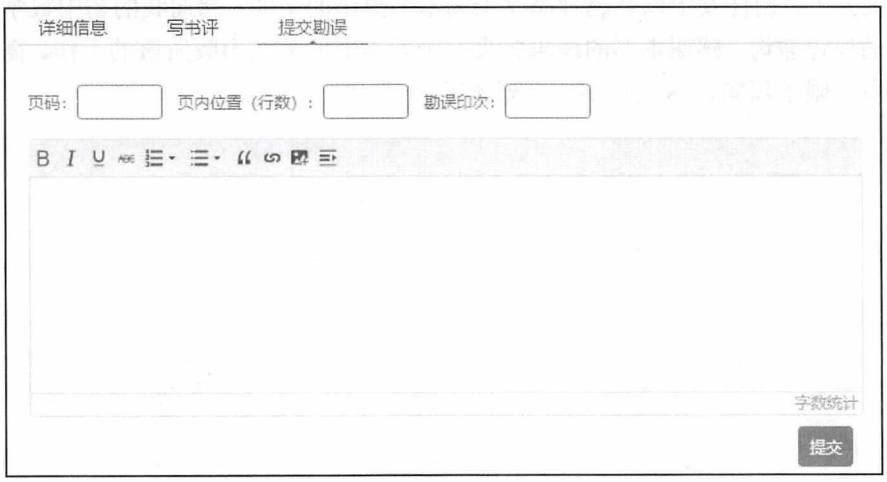
本书源代码请到异步社区的本书购买页面中下载。

请在异步社区本书页面中点击 配套资源 ，跳转到下载界面，按提示进行操作即可。注意：为保证正常购书用户的权益，会要求您输入提取码进行验证。

提交勘误

作者和编辑尽最大努力来确保书中内容的准确性，但难免还会存在差错。欢迎您将发现的问题告诉我们，帮助我们提升图书的质量。

当您发现错误时，请登录异步社区主页 https://www.epubit.com/，搜索到本书页面，点击“提交勘误”，输入勘误信息，最后单击“提交”按钮即可。之后本书的作者和编辑会对您提交的勘误进行审核。确认并接受后，您将获赠异步社区的 100 积分。积分可用于在社区兑换优惠券，以及兑换样书或奖品之用。



仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

扫码关注本书

请扫描下方二维码关注本书，即可在异步社区微信服务号中看到本书和进一步的服务信息。

与我们联系

我们的联系邮箱是 contact@epubit.com.cn。

如果您对本书有任何疑问或建议，请发邮件到此邮箱，邮件标题中请注明本书书名。

如果您有兴趣出版图书、录制教学视频，或参与图书翻译、技术审校等工作，可以发邮件，有意出版图书的作者还可以到异步社区在线提交投稿：

www.epubit.com/selfpublish/submission

如果您是学校、培训机构或企业，想批量购买本书或异步社区出版的其他图书，请发邮件联系我们。

如果您在网上发现有针对异步图书的各种形式的盗版行为，包括图书或部分内容的非授权传播，请您将怀疑有侵权行为的链接发邮件给我们。您的举动是对作者权益的保护，我们也由此才能继续为您带来有价值的内容。

关于异步社区和异步图书

异步社区是人民邮电出版社旗下 IT 专业图书社区，致力于出版精品 IT 技术图书和相关学习产品，为作译者提供优质出版服务，社区创办于 2015 年 8 月，提供超过 1000 种图书、近 1000 种电子书，以及众多技术文章和视频课程。更多详情请访问异步社区官网 https://www.epubit.com。

异步图书是由异步社区编辑团队策划出版的精品 IT 专业图书品牌，依托于人民邮电出版社近 30 年的计算机图书出版积累和专业编辑团队，在封面上印有异步图书的 LOGO。我们的出版领域包括软件开发、大数据、AI、测试、前端、网络技术等。

异步社区　　微信服务号

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

目 录

第 1 章 数据科学概述 …… 1
1.1 挑战 …… 2
1.1.1 工程实现的挑战 …… 2
1.1.2 模型搭建的挑战 …… 3
1.2 机器学习 …… 5
1.2.1 机器学习与传统编程 …… 5
1.2.2 监督式学习和非监督式学习 …… 8
1.3 统计模型 …… 8
1.4 关于本书 …… 10
第 2 章 Python 安装指南与简介：告别空谈 …… 12
2.1 Python 简介 …… 13
2.1.1 什么是 Python …… 15
2.1.2 Python 在数据科学中的地位 …… 16
2.1.3 不可能绕过的第三方库 …… 17
2.2 Python 安装 …… 17
2.2.1 Windows 下的安装 …… 18
2.2.2 Mac 下的安装 …… 21
2.2.3 Linux 下的安装 …… 24
2.3 Python 上手实践 …… 26
2.3.1 Python shell …… 26
2.3.2 第一个 Python 程序：Word Count …… 28
2.3.3 Python 编程基础 …… 30
2.3.4 Python 的工程结构 …… 34
2.4 本章小结 …… 35
第 3 章 数学基础：恼人但又不可或缺的知识 …… 36
3.1 矩阵和向量空间 …… 37
3.1.1 标量、向量与矩阵 …… 37

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

3.1.2 特殊矩阵 …… 39
3.1.3 矩阵运算 …… 39
3.1.4 代码实现 …… 42
3.1.5 向量空间 …… 44
3.2 概率：量化随机 …… 46
3.2.1 定义概率：事件和概率空间 …… 47
3.2.2 条件概率：信息的价值 …… 48
3.2.3 随机变量：两种不同的随机 …… 50
3.2.4 正态分布：殊途同归 …… 52
3.2.5 P-value：自信的猜测 …… 53
3.3 微积分 …… 55
3.3.1 导数和积分：位置、速度 …… 55
3.3.2 极限：变化的终点 …… 57
3.3.3 复合函数：链式法则 …… 58
3.3.4 多元函数：偏导数 …… 59
3.3.5 极值与最值：最优选择 …… 59
3.4 本章小结 …… 61
第 4 章 线性回归：模型之母 …… 62
4.1 一个简单的例子 …… 64
4.1.1 从机器学习的角度看这个问题 …… 66
4.1.2 从统计学的角度看这个问题 …… 69
4.2 上手实践：模型实现 …… 73
4.2.1 机器学习代码实现 …… 74
4.2.2 统计方法代码实现 …… 77
4.3 模型陷阱 …… 82
4.3.1 过度拟合：模型越复杂越好吗 …… 84
4.3.2 模型幻觉之统计学方案：假设检验 …… 87
4.3.3 模型幻觉之机器学习方案：惩罚项 …… 89
4.3.4 比较两种方案 …… 92
4.4 模型持久化 …… 92
4.4.1 模型的生命周期 …… 93
4.4.2 保存模型 …… 93
4.5 本章小结 …… 96

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

第 5 章　逻辑回归：隐藏因子 …… 97
5.1　二元分类问题：是与否 …… 98
5.1.1　线性回归：为何失效 …… 98
5.1.2　窗口效应：看不见的才是关键 …… 100
5.1.3　逻辑分布：胜者生存 …… 102
5.1.4　参数估计之似然函数：统计学角度 …… 104
5.1.5　参数估计之损失函数：机器学习角度 …… 104
5.1.6　参数估计之最终预测：从概率到选择 …… 106
5.1.7　空间变换：非线性到线性 …… 106
5.2　上手实践：模型实现 …… 108
5.2.1　初步分析数据：直观印象 …… 108
5.2.2　搭建模型 …… 113
5.2.3　理解模型结果 …… 116
5.3　评估模型效果：孰优孰劣 …… 118
5.3.1　查准率与查全率 …… 119
5.3.2　ROC 曲线与 AUC …… 123
5.4　多元分类问题：超越是与否 …… 127
5.4.1　多元逻辑回归：逻辑分布的威力 …… 128
5.4.2　One-vs.-all：从二元到多元 …… 129
5.4.3　模型实现 …… 130
5.5　非均衡数据集 …… 132
5.5.1　准确度悖论 …… 132
5.5.2　一个例子 …… 133
5.5.3　解决方法 …… 135
5.6　本章小结 …… 136
第 6 章　工程实现：计算机是怎么算的 …… 138
6.1　算法思路：模拟滚动 …… 139
6.2　数值求解：梯度下降法 …… 141
6.3　上手实践：代码实现 …… 142
6.3.1　TensorFlow 基础 …… 143
6.3.2　定义模型 …… 148
6.3.3　梯度下降 …… 149
6.3.4　分析运行细节 …… 150

书籍是人类进步的阶梯　仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

6.4 更优化的算法：随机梯度下降法 …… 153
6.4.1 算法细节 …… 153
6.4.2 代码实现 …… 154
6.4.3 两种算法比较 …… 156
6.5 本章小结 …… 158
第 7 章 计量经济学的启示：他山之石 …… 159
7.1 定量与定性：变量的数学运算合理吗 …… 161
7.2 定性变量的处理 …… 162
7.2.1 虚拟变量 …… 162
7.2.2 上手实践：代码实现 …… 164
7.2.3 从定性变量到定量变量 …… 168
7.3 定量变量的处理 …… 170
7.3.1 定量变量转换为定性变量 …… 171
7.3.2 上手实践：代码实现 …… 171
7.3.3 基于卡方检验的方法 …… 173
7.4 显著性 …… 175
7.5 多重共线性：多变量的烦恼 …… 176
7.5.1 多重共线性效应 …… 176
7.5.2 检测多重共线性 …… 180
7.5.3 解决方法 …… 185
7.5.4 虚拟变量陷阱 …… 188
7.6 内生性：变化来自何处 …… 191
7.6.1 来源 …… 192
7.6.2 内生性效应 …… 193
7.6.3 工具变量 …… 195
7.6.4 逻辑回归的内生性 …… 198
7.6.5 模型的联结 …… 200
7.7 本章小结 …… 201
第 8 章 监督式学习：目标明确 …… 202
8.1 支持向量学习机 …… 203
8.1.1 直观例子 …… 204
8.1.2 用数学理解直观 …… 205
8.1.3 从几何直观到最优化问题 …… 207

书籍是人类进步的阶梯

8.1.4 损失项 ····· 209
8.1.5 损失函数与惩罚项 ····· 210
8.1.6 Hard margin 与 soft margin 比较 ····· 211
8.1.7 支持向量学习机与逻辑回归：隐藏的假设 ····· 213
8.2 核函数 ····· 216
8.2.1 空间变换：从非线性到线性 ····· 216
8.2.2 拉格朗日对偶 ····· 218
8.2.3 支持向量 ····· 220
8.2.4 核函数的定义：优化运算 ····· 221
8.2.5 常用的核函数 ····· 222
8.2.6 Scale variant ····· 225
8.3 决策树 ····· 227
8.3.1 决策规则 ····· 227
8.3.2 评判标准 ····· 229
8.3.3 代码实现 ····· 231
8.3.4 决策树预测算法以及模型的联结 ····· 231
8.3.5 剪枝 ····· 235
8.4 树的集成 ····· 238
8.4.1 随机森林 ····· 238
8.4.2 Random forest embedding ····· 239
8.4.3 GBTs 之梯度提升 ····· 241
8.4.4 GBTs 之算法细节 ····· 242
8.5 本章小结 ····· 244
第 9 章 生成式模型：量化信息的价值 ····· 246
9.1 贝叶斯框架 ····· 248
9.1.1 蒙提霍尔问题 ····· 248
9.1.2 条件概率 ····· 249
9.1.3 先验概率与后验概率 ····· 251
9.1.4 参数估计与预测公式 ····· 251
9.1.5 贝叶斯学派与频率学派 ····· 252
9.2 朴素贝叶斯 ····· 254
9.2.1 特征提取：文字到数字 ····· 254
9.2.2 伯努利模型 ····· 256

书籍是人类进步的阶梯

9.2.3 多项式模型 …… 258
9.2.4 TF-IDF …… 259
9.2.5 文本分类的代码实现 …… 260
9.2.6 模型的联结 …… 265
9.3 判别分析 …… 266
9.3.1 线性判别分析 …… 267
9.3.2 线性判别分析与逻辑回归比较 …… 269
9.3.3 数据降维 …… 270
9.3.4 代码实现 …… 273
9.3.5 二次判别分析 …… 275
9.4 隐马尔可夫模型 …… 276
9.4.1 一个简单的例子 …… 276
9.4.2 马尔可夫链 …… 278
9.4.3 模型架构 …… 279
9.4.4 中文分词：监督式学习 …… 280
9.4.5 中文分词之代码实现 …… 282
9.4.6 股票市场：非监督式学习 …… 284
9.4.7 股票市场之代码实现 …… 286
9.5 本章小结 …… 289
第 10 章 非监督式学习：聚类与降维 …… 290
10.1 K-means …… 292
10.1.1 模型原理 …… 292
10.1.2 收敛过程 …… 293
10.1.3 如何选择聚类个数 …… 295
10.1.4 应用示例 …… 297
10.2 其他聚类模型 …… 298
10.2.1 混合高斯之模型原理 …… 299
10.2.2 混合高斯之模型实现 …… 300
10.2.3 谱聚类之聚类结果 …… 303
10.2.4 谱聚类之模型原理 …… 304
10.2.5 谱聚类之图片分割 …… 307
10.3 Pipeline …… 308
10.4 主成分分析 …… 309

书籍是人类进步的阶梯

10.4.1 模型原理 ····· 310
10.4.2 模型实现 ····· 312
10.4.3 核函数 ····· 313
10.4.4 Kernel PCA 的数学原理 ····· 315
10.4.5 应用示例 ····· 316
10.5 奇异值分解 ····· 317
10.5.1 定义 ····· 317
10.5.2 截断奇异值分解 ····· 317
10.5.3 潜在语义分析 ····· 318
10.5.4 大型推荐系统 ····· 320
10.6 本章小结 ····· 323
第 11 章 分布式机器学习：集体力量 ····· 325
11.1 Spark 简介 ····· 327
11.1.1 Spark 安装 ····· 328
11.1.2 从 MapReduce 到 Spark ····· 333
11.1.3 运行 Spark ····· 335
11.1.4 Spark DataFrame ····· 336
11.1.5 Spark 的运行架构 ····· 339
11.2 最优化问题的分布式解法 ····· 341
11.2.1 分布式机器学习的原理 ····· 341
11.2.2 一个简单的例子 ····· 342
11.3 大数据模型的两个维度 ····· 344
11.3.1 数据量维度 ····· 344
11.3.2 模型数量维度 ····· 346
11.4 开源工具的另一面 ····· 348
11.4.1 一个简单的例子 ····· 349
11.4.2 开源工具的阿喀琉斯之踵 ····· 351
11.5 本章小结 ····· 351
第 12 章 神经网络：模拟人的大脑 ····· 353
12.1 神经元 ····· 355
12.1.1 神经元模型 ····· 355
12.1.2 Sigmoid 神经元与二元逻辑回归 ····· 356
12.1.3 Softmax 函数与多元逻辑回归 ····· 358

书籍是人类进步的阶梯

12.2 神经网络 …… 360
12.2.1 图形表示 …… 360
12.2.2 数学基础 …… 361
12.2.3 分类例子 …… 363
12.2.4 代码实现 …… 365
12.2.5 模型的联结 …… 369
12.3 反向传播算法 …… 370
12.3.1 随机梯度下降法回顾 …… 370
12.3.2 数学推导 …… 371
12.3.3 算法步骤 …… 373
12.4 提高神经网络的学习效率 …… 373
12.4.1 学习的原理 …… 373
12.4.2 激活函数的改进 …… 375
12.4.3 参数初始化 …… 378
12.4.4 不稳定的梯度 …… 380
12.5 本章小结 …… 381
第 13 章 深度学习：继续探索 …… 383
13.1 利用神经网络识别数字 …… 384
13.1.1 搭建模型 …… 384
13.1.2 防止过拟合之惩罚项 …… 386
13.1.3 防止过拟合之 dropout …… 387
13.1.4 代码实现 …… 389
13.2 卷积神经网络 …… 394
13.2.1 模型结构之卷积层 …… 395
13.2.2 模型结构之池化层 …… 397
13.2.3 模型结构之完整结构 …… 399
13.2.4 代码实现 …… 400
13.2.5 结构真的那么重要吗 …… 405
13.3 其他深度学习模型 …… 406
13.3.1 递归神经网络 …… 406
13.3.2 长短期记忆 …… 407
13.3.3 非监督式学习 …… 409
13.4 本章小结 …… 411

书籍是人类进步的阶梯

第1章

数据科学概述

The purpose of computing is insight, not numbers.

（计算的目的不在于数字，而在于洞察事物。）

——*Richard Hamming*

1.1 挑战

1.2 机器学习

1.3 统计模型

1.4 关于本书

书籍是人类进步的阶梯

随着云计算和人工智能的发展，数据科学这门新的综合学科被越来越多的人所熟知，业界也普遍看好其在未来的发展前景。体现在就业市场上，与这个行业相关的数据科学家和数据工程师[1]成为了“21 世纪最吸引人的职业”[2]。

就像“一千个人眼里有一千个哈姆雷特”一样，对于什么是数据科学也有很多种不同的解读，并由此衍生出很多相关概念，比如数据驱动（data driven）、大数据（big data）、分布式计算（parallel computing）等。这些概念虽然各有侧重点，但它们都毫无争议地围绕着同一个主题：如何从实际的生活中提取出数据，然后利用计算机的运算能力和模型算法从这些数据中找出一些有价值的内容，为商业决策提供支持。这正是数据科学的核心内涵。

传统的数据分析手段是所谓的商业智能（business intelligence）。这种方法通常将数据按不同的维度交叉分组，并在此基础上，利用统计方法分析每个组别里的信息。比如商业智能中最常见的问题是：“过去 3 个月，通过搜索引擎进入网站并成功完成注册的新用户里，年龄分布情况如何？若将上面的用户群按年龄段分组，各组中有多大比例的用户在完成注册后，完成了至少一次消费？”

这样的分析是非常有用的，能揭示一些数据的直观信息。但这样的方法如同盲人摸象，只能告诉我们数据在某个局部的情况，而不能给出数据的全貌。而且对于某些问题，这样的结果显得有些不够用。比如用户注册之后完成消费的比例与哪些因素相关？又比如对于某个客户，应该向他推荐什么样的商品？在这些场景下，我们就需要更加精细的数据分析工具——机器学习和统计模型。在我看来，这些内容是数据科学的核心内容，也是本书介绍的重点。

1.1 挑战

在数据科学实践中，我们将使用较为复杂的机器学习或统计模型对数据做精细化的分析和预测。这在工程实现和模型搭建两方面都提出了挑战，如图 1-1 所示。

1.1.1 工程实现的挑战

数据科学在工程上的挑战可以大致分为 3 类：特征提取、矩阵运算和分布式机器学习。

（1）一个建模项目的成功在很大程度上依赖于建模前期的特征提取。它包含数据清

[1] 数据科学家（data scientist）的主要工作是为数据搭建模型，要求具有扎实的数学以及统计知识；数据工程师（data engineer）的主要工作是利用计算机分析数据和实现数据科学家设计好的模型，要求具有良好的编程实现能力。实际上这两种职业的工作内容有很多重叠的部分，因此要求从业者同时具备较好的数理知识和工程实现能力。

[2] Davenport T H, Patil D J. Data Scientist: The Sexiest Job of the 21st Century[J]. Harvard Business Review, 2012, 90(10):70-76.

书籍是人类进步的阶梯

洗、数据整合、变量归一化等。经过处理后，原本搅作一团的原始数据将被转换为能被模型使用的特征。这些工作需要大量的自动化程序来处理，特别是面对大数据时，因为这些大数据无法靠“人眼”来检查。在一个典型的建模项目中，这部分花费的时间远远大于选择和编写模型算法的时间。

（2）对于一个复杂的数学模型，计算机通常需要使用类似随机梯度下降法的最优化算法来估算它的模型参数。这个过程需要大量的循环，才能使参数到达收敛值附近。因此即使面对的是很小的数据集，复杂的模型也需要很长时间才能得到正确的参数估计。而且模型在结构上越复杂，需要估计的参数也就越多。对这些大量的模型参数同时做更新，在数学上对应着矩阵运算。但传统的 CPU 架构并不擅长做这样的运算，这导致模型训练需要耗费大量的时间。为了提高模型的训练速度，需要将相应的矩阵运算（模型参数的估算过程）移植到 GPU 或者特制的计算芯片上，比如 TPU。

（3）近年来，随着分布式系统的流行和普及，存储海量数据成为了业界的标配。为了能在这海量的数据上使用复杂模型，需要将原本在一台机器上运行的模型算法改写成能在多台机器上并行运行，这也是分布式机器学习的核心内容。

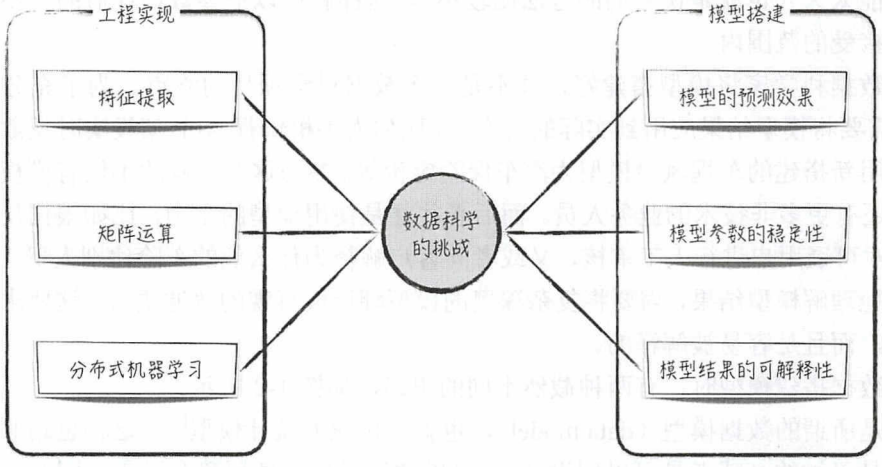


图 1-1

1.1.2 模型搭建的挑战

数据科学对模型搭建的要求也可以总结为 3 点：模型预测效果好、模型参数是稳定且“正确”的、模型结果容易解释。

（1）模型的预测效果好，这是数据科学成功的关键。而一个模型的预测效果取决于它的假设是否被满足。从数学上来看，任何一个模型除去假设部分，它的其他推导都是

书籍是人类进步的阶梯

严谨的数学演算，是无懈可击的。因此模型假设就像模型的阿喀琉斯之踵[3]，是它唯一的薄弱环节。当问题场景或数据满足模型假设时，模型的效果一定不会差，反之，则预测效果就无法保证了。但在实际生产中，针对一个具体的问题，几乎不可能找到一个模型，它的假设被百分之百地满足。这时就需要避重就轻，通过特征提取等手段，尽量避免违反那些对结果影响很大的假设。这就是为什么说“所有模型都是错的，但是，其中有一些是有用的”[4]。

（2）除了被用来对未知数据做预测外，模型另一个重要的功能就是对已有数据做分析，比如哪个变量对结果的影响最大或者某个变量对结果到底是正向影响还是负向影响等。这些分析结果在很大程度上依赖于模型参数的估计值，后者的准确与否直接决定分析结果的质量。但问题是，模型参数的估计值是不太“可靠”的。例如从训练数据中随机抽取两个不完全一样的数据子集 A 和 B，然后用这两个数据集分别训练同一个模型。得到的参数估计值几乎不可能完全一样。从数学的角度来看，这说明模型参数的估计值其实是一个随机变量，具体的值取决于训练模型时使用的数据。于是我们要求这些估计值是“正确”的：围绕参数真实值上下波动（也就是说它们的期望等于参数真实值）。我们还要求这些估计值是稳定的：波动的幅度不能太大（也就是说它们的方法比较小）。这样就可以把参数估计值的“不可靠性”控制在可接受的范围内。

（3）数据科学家将模型搭建好，并不是一个数据科学项目的终点。为了充分发挥数据的价值，需要将模型结果应用到实际的生产中，比如为手机银行 APP 架设实时反欺诈系统，或者将利用新搭建的车祸风险模型为汽车保险定价等。参与这个过程的不仅有懂模型的数据科学家，还有更多非技术的业务人员。而后者往往是使用模型的主力，比如根据反欺诈系统的结果，对可疑用户进行人工审核，又或者向客户解释为什么他的车险比别人贵。为了帮助他们更好地理解模型结果，需要将复杂深奥的模型翻译成直观的普通语言。这要求模型是能被解释的，而且是容易被解释的。

在对数据搭建模型时，有两种截然不同的思路，如图 1-2 所示。

一种是所谓的数据模型（data model），也就是传统的统计模型[5]。这种思路假设数据的产生过程是已知的（或者是可以假设的），可以通过模型去理解整个过程。因此，这类模型通常具有很好的可解释性，分析其稳定性的数学工具也很多，能很好地满足上面提到的后两点。但是在实际生产中，这些模型的预测效果并不好，或者更准确地说，单独使用时，预测效果并不理想。

---

[3] 阿喀琉斯是希腊神话中的一个人物。除了未被冥河浸泡过的脚后跟外，他全身近乎刀枪不入，有“希腊第一勇士”之称。在特洛伊战争中，阿喀琉斯被箭射中脚后跟而死去。因此阿喀琉斯之踵欲指某事物的最大或者唯一弱点。

[4] 出自英国统计学家 George Edward Pelham Box。

[5] Breiman L. [Statistical Modeling: The Two Cultures]: Rejoinder[J]. Statistical Science, 2001, 16(3):199-231.

书籍是人类进步的阶梯

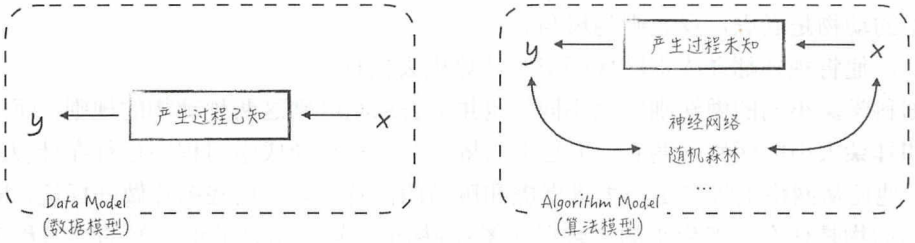


图 1-2

另一种是所谓的算法模型（algorithm model），也就是机器学习。这类模型是人工智能的核心内容，它们假设数据的产生过程是复杂且未知的。建模的目的是尽可能地从结构上“模仿”数据的产生过程，从而达到较好的预测效果。但代价是模型的可解释性很差，而且模型稳定性的分析方法也不多。

正如上面的分析，统计学和机器学习在某些方面具有极好的互补性。因此在实际的生产中，为了将一个数据科学项目做得尽可能完美，我们需要将这两种思路结合起来使用。比如使用机器学习的模型对数据建模，然后借鉴数据模型的分析工具，分析模型的稳定性和给出模型结果的直观解释。

1.2 机器学习

在讨论机器学习之前，我们先来总结一下人是怎么学习的：在面对一个具体的问题时，人首先会根据已有的经验和当前的信息做出反应行动，然后按照行动获得的反馈去修正自己的经验，并不断重复这个过程。

机器学习就是通过程序让计算机“学会”人的学习过程。换句话说，机器学习是一个计算机程序，这个程序能够根据“经验”自我完善。那么问题来了，既然都是程序，机器学习和传统编程有什么不同呢？

1.2.1 机器学习与传统编程

首先，通过一个假想的例子来体会这两者之间的差异。假设我们需要构建一个能区分老虎和斑马图片的系统，而这个任务被交给了程序员小李和数据科学家小胡。

程序员小李拿到这个任务之后，他首先总结了两条分辨这两种动物的经验：

- 斑马的毛发是黑白相间的，而老虎的毛发有 3 种颜色：黑色、白色和黄色。
- 斑马的耳朵比较大，而老虎的耳朵比较小。

然后，小李将上面的两条经验总结成规则：如果图中的动物有黄色毛发或者耳朵比较小，

书籍是人类进步的阶梯

那么图中的动物是老虎，反之则为斑马。

最后，他将规则翻译成程序代码交给计算机去执行。

数据科学家小胡的做法则完全不同。他并不去总结区别这两种动物的规则，而是假设毛发颜色和耳朵大小能区别这两者。于是小胡将这个逻辑翻译成学习程序运行在计算机上。与此同时，他还从网络上收集了一大堆老虎和斑马的图片，并将这些图片做好标记，即每张图片对应的动物是什么。然后小胡将数据（图片+标记）输入给之前编写好的学习程序，而后者就可以根据数据不断累积经验和总结规则，这个过程被称为模型训练。经过一段时间的训练后，学习程序最后得到了一个模型（以代码的形式存在）。这个得到的模型和小李自己编写的程序一样，可以用来区分老虎和斑马的图片。

小李和小胡各自的工作流程如图 1-3 所示。从编程的角度来看，机器学习是一种能自动生成程序的特殊程序。

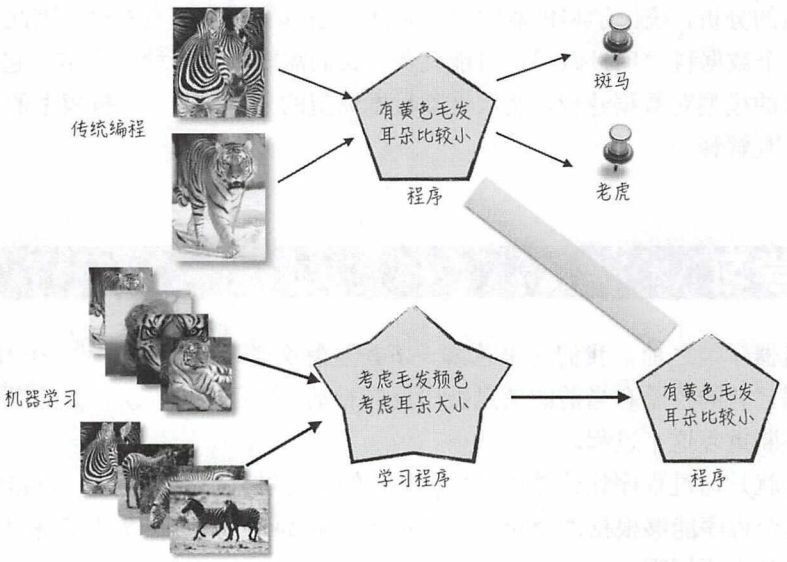


图 1-3

也许上面的例子有点过于抽象了，下面再来看一个简单而具体的例子。假设我们想通过一个成年人的身高（用变量x表示）来预测他的体重（用变量y表示）。这项工作同样被交给了小李和小胡。

小李拿到问题之后，首先上网查阅了成年人身高与体重的相关研究资料。他发现这两者的关系可以用如下的公式表示。

$$y = 0.9x - 90 \tag{1-1}$$

接着他将公式（1-1）转换为程序里面的函数，这个函数的输入参数是身高，输出是预

书籍是人类进步的阶梯

测的体重。

小胡接到任务后，首先假设身高和体重的关系可以用线性回归模型来表示，也就是公式（1-2），其中a和b为未知的模型参数

$$y = ax + b \tag{1-2}$$

同图片识别中的做法类似，小胡接下来收集了大量的人体身高和体重的数据。他将这些数据输入给模型，而模型将根据得到的数据估计未知参数a, b。估计的原则是使得模型预测值与实际值的差距达到最小。经过计算之后，得到参数的估计值为$\hat{a} = 0.8, \hat{b} = -100$。也就是说经过模型训练后，小胡得到了如公式（1-3）所示的程序

$$y = 0.8x - 100 \tag{1-3}$$

图 1-4 所示是小李和小胡各自提供的解决方案。

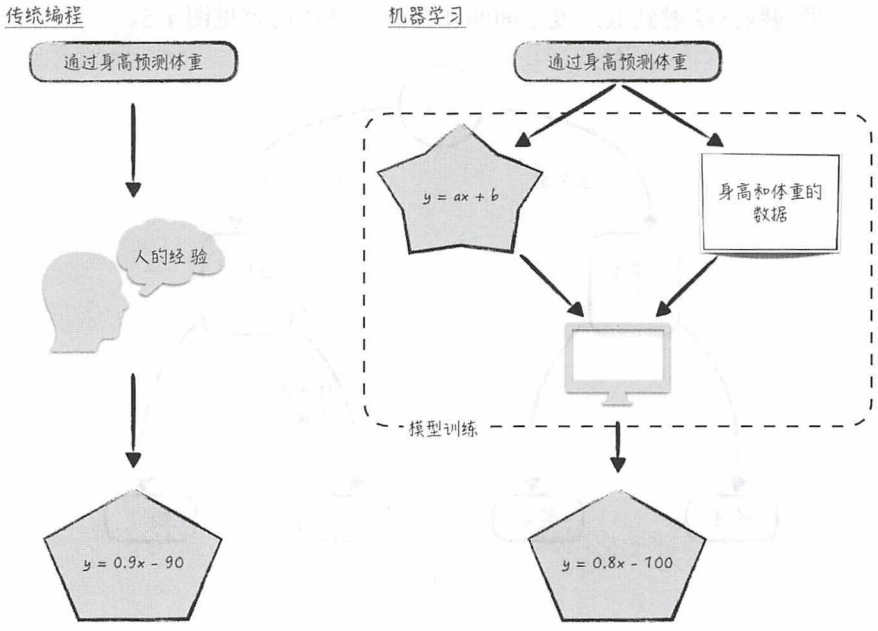


图 1-4

总结一下，传统的编程方式是人类自己积累经验，并将这些经验转换为规则或数学公式，然后就是用编程语言去表示这些规则和公式。而机器学习可以被看作一种全新的编程方式。在进行机器学习时，人类不需要总结具体的规则或公式，只需制订学习的步骤，然后将大量的数据输入给计算机。后者可以根据数据和人类提供的学习步骤自己总结经验，自动升级。计算机“学习”完成之后会得到一个模型程序，而这个由程序生成的程序可以达到甚至超过人类自身的水平。

书籍是人类进步的阶梯

1.2.2 监督式学习和非监督式学习

机器学习根据所用的训练数据可以分为两类。一类是监督式学习，这类模型的特点是训练数据里有标注，也就是常说的被预测量y。1.2.1 节里讨论的两个例子都是监督式学习。监督式学习按标注的类型又可以细分为两类：分类和回归。如果数据里的标注表示事物的类别，也就是说标注是离散的，那么相应的模型就属于分类，比如 1.2.1 节里动物图片识别的例子。如果标注表示具体的数量，也就是说标注的是连续的，那么相应的模型就属于回归，比如 1.2.1 节里通过身高预测体重的例子。

另一类是非监督式学习，这类模型所用的训练数据里并没有标注，只有自变量x。非监督式学习根据用途又可以分为两类：聚类和降维。把“距离”相近的点归于一类叫作聚类，而将高维空间里的数据映射到低维度空间叫作降维，具体内容见图 1-5。

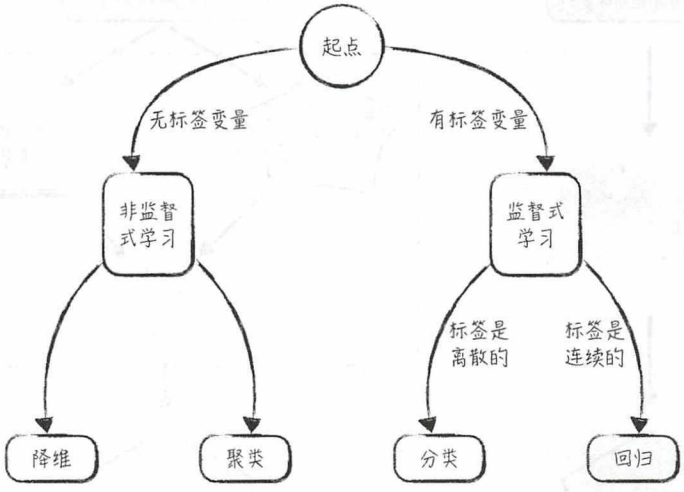


图 1-5

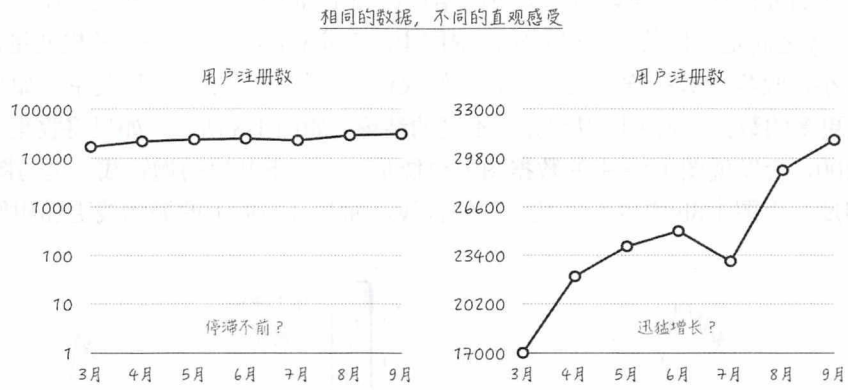
1.3 统计模型

从上面的例子中可以看到，机器学习非常依赖所用的训练数据。但是数据就百分之百可靠吗？下面就来看两个数据“说谎”的例子。

如图 1-6 所示，我们将某 APP 每月的用户注册数表示在图中。图 1-6a 给人的直观印象是每月的安装数是大致差不多的，没有明显的增长。而图 1-6b 给人不同的印象，从 3 月份开始，用户注册数大幅度增长。但其实两幅图的数据是一模一样的，给人不同的感觉是因为

书籍是人类进步的阶梯

图 1-6a 中纵轴的起点是 0，而且使用了对数尺度；而图 1-6b 的纵轴是从 17 000 开始的，而且使用的是线性尺度。



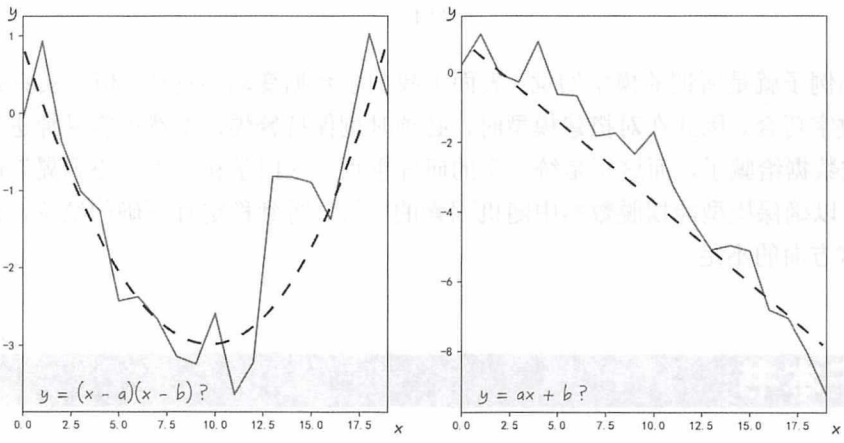
（a） （b）

图 1-6

读者可能会觉得上面这个例子太过简单了，只需要使用一些简单的统计指标，比如平均值或每个月的增长率，就可以避免错误的结论。那么下面来看一个复杂一点的例子。

当得到如图 1-7 所示的两组数据时，我们应该如何用模型去描述数据的变化规律呢？

- 对于图 1-7a，数据的图形有点像抛物线，因此选择二次多项式拟合是一个比较合理的选择。于是假设模型的形式为$y = (x - a)(x - b)$。然后使用数据去估计模型中的未知参数a, b。得到的结果还不错，模型的预测值与真实值的差异并不大。



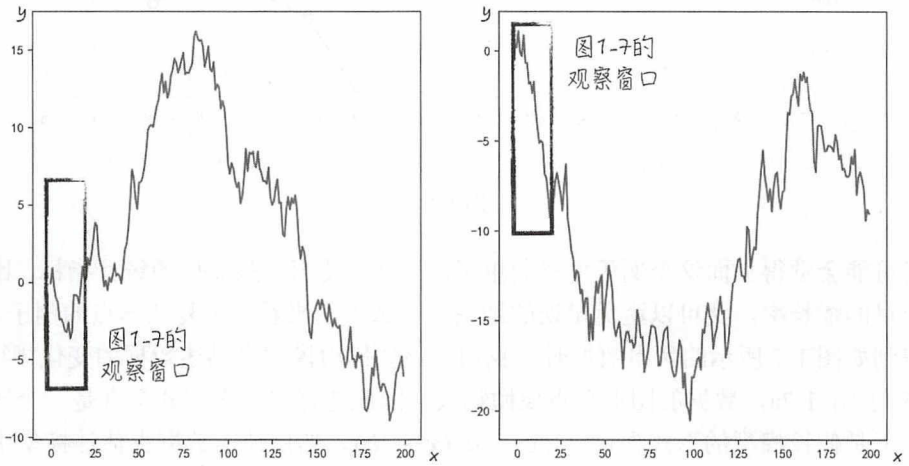
（a） （b）

图 1-7

书籍是人类进步的阶梯

- 对于图 1-7b，数据之间有明显的线性关系，所以使用线性回归对其建模，即 $y = ax + b$。与上面类似，得到的模型结果也不错。

根据上面的分析结果，可以得出如下的结论，图 1-7a 中的x与y之间是二次函数关系，而图 1-7b 的x与y之间是线性关系。但其实两幅图中的变量y都是与x无关的随机变量，只是因为观察窗口较小，收集的数据样本太少，让我们误以为它们之间存在某种关系。如果增大观察窗口，收集更多的数据，则可以得到完全不同的结论。如图 1-8 所示，如果将收集的样本数从 20 增加到 200，会发现图 1-8a 中的数据图形更像是一个向下开口的抛物线，这与图 1-7a 中的结论完全相反。而图 1-8b 中也不再是向下的直线，而与开口向上的抛物线更加相似。



(a) (b)

图 1-8

上面的例子就是所谓的模型幻觉：表面上找到了数据变动的规律，但其实只是由随机扰动引起的数字巧合。因此在对搭建模型时，必须时刻保持警惕，不然很容易掉进数据的“陷阱”里，被数据给骗了，而这正是统计学的研究重点。这门学科会“小心翼翼”地处理它的各种模型，以确保模型能摆脱数据中随机因素的干扰，得到稳定且正确的结论，正好弥补机器学习在这方面的不足。

1.4 关于本书

数据科学涉及计算机编程和数学建模这两个方面。它们之间的交集并不多，所强调的技能也有很大区别。这体现在实际生产中就是懂模型的人不懂编程，懂编程的人不懂模型，两

书籍是人类进步的阶梯

书籍是人类进步的阶梯

者兼备的人才非常稀缺。本书的第一个目的就是将这两者的鸿沟弥补起来，注重模型假设和数学推导的同时，强调如何用代码实现模型。

- 从模型之间的联系和区别出发，分析各个模型的优缺点。帮助非数学专业的读者更加深入地理解模型的假设和适用范围，而不只是停留在会使用开源模型库的 API。
- 通过大量实际案例和代码展示，帮助非计算机专业的读者能独立上机实践模型算法，而不只停留在模型的理论研究。

对于数据科学中的模型搭建，统计学和机器学习是其最重要的组成部分。这两门学科的侧重点并不相同，在很多方面它们是彼此很好的补充。在面对一个实际问题时，若能将两者的方法相结合，能更好地挖掘数据的内在规律，从而更大程度地发挥数据的价值。这是本书的第二个目的。

- 将机器学习和统计结合起来，并借鉴统计学在经济领域的应用，为机器学习的算法提供一个生动而又不失精确的解释。同时用丰富的图片将这些解释直观地表现出来，帮助专业人员将模型和算法解释给非专业的业务人员，推动模型的落地和应用。
- 借鉴计量经济学的方法，深入探讨模型应用中常常被人们（特别是机器学习专业人员）忽略的问题，如模型是否稳定、模型结果是否可靠等，帮助读者反思建模过程中是否有考虑不周到的地方，以至于模型得到错误的结论。

当前，数据科学有两个最热门的前沿领域：分布式机器学习和深度学习。本书有专门的章节讨论它们，展示这两个领域想要解决的问题和目前最好（或最流行）的解决方案。这是本书的第三个目的：从宏观的角度向读者展示什么是数据科学，想要解决的问题、主要的方法以及未来的发展方向。

本书并不试图成为机器学习或统计学的参考书。在之后的章节里，有关模型的数学推导都只是简略讲解，并不做详细证明（本书也不会为了迎合行文简便，一味地回避这些难点[6]）。本书讨论的重点是数据科学的整个工作流程（Pipeline）：不止是搭建模型、用数据去训练模型，而是如何对数据进行预处理，初步分析数据、搭建并评估模型以及根据结果分析模型的缺点进而改进模型。

[6] 伟大的法国天才数学家，抽象代数的奠基人，埃瓦里斯特·伽罗瓦（Évariste Galois）曾说过："一个作家对读者做的最大的恶就是隐藏难点（un auteur ne nuit jamais tant à ses lecteurs que quand il dissimule une difficulté）。"这也是本书在编写时遵循的原则之一：直面数据科学领域里的难点，但用通俗的语言讲解它们。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

第2章

Python安装指南与简介：告别空谈

If you are immune to boredom, there is literally nothing you cannot accomplish.

（如果你能忍耐乏味，你将无往不利。）

——*David Foster Wallace*

2.1 Python 简介

2.2 Python 安装

2.3 Python 上手实践

2.4 本章小结

书籍是人类进步的阶梯

书籍是人类进步的阶梯

本章介绍数据科学中最常用的 IT 工具：Python 以及相关第三方库。本书后面章节的代码示例和模型实现都基于此。正所谓“工欲善其事，必先利其器”，为了更深入地理解数据科学，读者需要确保已安装好所需的 IT 工具，并基本熟悉其使用方法。

如果读者对此已十分熟悉，可选择跳过本章。虽然如此，作者仍推荐阅读本章。

2.1 Python 简介

记得在几年前，数据科学刚刚兴起的时候，我有幸参加了一次行业里的聚会，探讨数据科学的内涵、机会以及难点等。其中有一页 PPT 给我留下了深刻的印象，如图 2-1 所示。

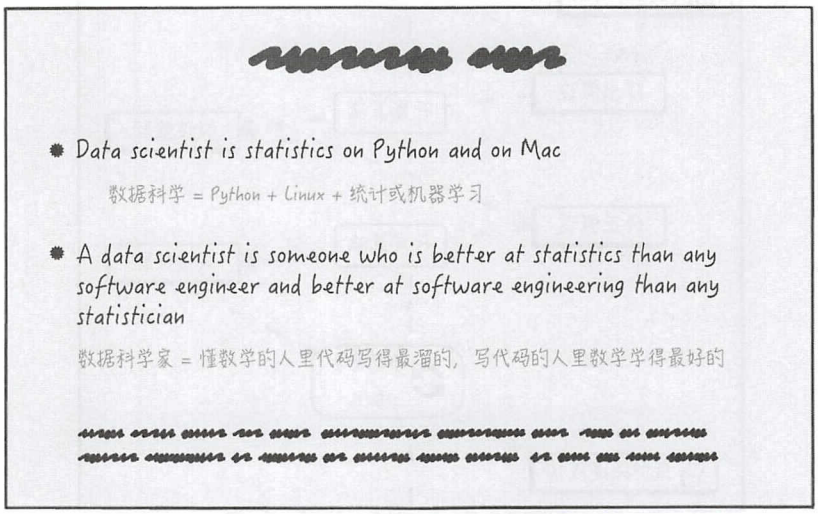


图 2-1

当时看到后一笑而过，程序员总会强调自己使用的语言是最好的，比如著名的“PHP 是世界上最好的语言”。这里用略带戏谑口吻的段子来强调 Python 语言的重要性，感觉很新颖。但是经过这几年的工作实践，我愈发感到：这虽然是个段子，却在很大程度上真实地描述了数据科学这个行业的工作状态，甚至可以说是直击痛点。

为了更形象地说明这个问题，我们不妨假设我们有个小伙伴叫小安。数学系毕业的她刚刚获得了一份数据科学相关的工作。上班的第一天，她满怀热情而且迫不及待地想接触这个崭新的行业。但是她很快就发现自己面临着一个巨大的困难：

工作所需要处理的数据并不存放在她的个人电脑里，而是都保存在远程服务器上，有的存放在传统的关系型数据库里，有的存放在 Hadoop 集群上。与个人电脑大多使用的 Windows

书籍是人类进步的阶梯

书籍是人类进步的阶梯

不同，远程服务器上使用的都是类 Linux 系统[1]。小安很不习惯这种操作系统，因为熟悉的图形化界面不见了。一切操作，比如最简单的读取文件都需要自己编程来实现。因此，小安很想找到一款书写简单、易学易用的编程语言。

更致命的是，小安所熟悉的数据建模软件，比如 SPSS、MATLAB 等在新的工作环境里都没办法使用了。而小安在平时的工作里会经常用到这些软件提供的一些基础的算法，比如线性回归、逻辑回归等。所以，她希望所找到的编程语言也有可以方便使用的算法库，当然最好是免费的。

小安的日常工作流程大致如图 2-2 所示。

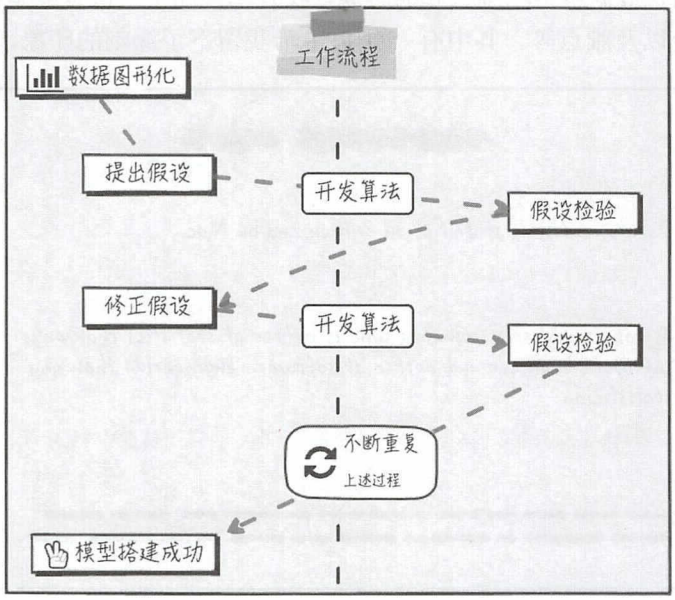


图 2-2

[1] 关系型数据库（Relational Database），用比较学术的语言表述：它是建立在关系模型基础上的数据库，借助于集合代数等数学概念和方法来处理数据库中的数据。关系型数据库从直观上来看，它用二维网状表来表示数据，有点类似于数学中的矩阵，每一行表示一个个体，每一列表示一个属性。 在实践中，常用的标准数据查询语言 SQL 来操作关系型数据库里的数据。比较常见的数据库有 Oracle、DB2、MSSQL 等。

Apache Hadoop 是一款开源的分布式数据系统。它包括资源管理、存储和计算 3 部分，支持在大型计算机集群上运行应用程序。目前，Hadoop 已形成一个强大的“生态圈”，是流行较广的大数据储存和处理软件。

Linux 是一款开源的操作系统，它的作用与 Windows 类似，是管理计算机硬件与软件资源的计算机程序。由于 Linux 的开放性，是目前使用最为广泛的操作系统，特别是远程服务器和其他大型平台上。比如世界上前 500 台速度最快的超级计算机 90%以上运行类 Linux 系统。

以上简介均参考自维基百科。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

整个过程和小安最爱的乒乓球很像，把假设当成“球”发给数据，再根据数据的“回球”，做出调整，并重复上述的动作。因此，小安又在她的要求里多加了一条：编程语言能随时修改随时使用，不需要编译。最好能有个即时响应的命令窗口，方便她快速验证自己的想法。

经过一番搜索，小安激动地告诉大家，她找到了满足她所有要求的 IT 工具：Python。下面我们就让小安来给我们介绍一下她的选择。

2.1.1 什么是 Python

Python（发音：/ˈpaɪθən/ ），是一种面向对象、解释型的计算机程序语言。它的语法简单，并且包含了一组功能完备的标准库，能够轻松完成很多常见的任务[2]。说起 Python，它的诞生也颇有意思。1989 年的圣诞假期，荷兰程序员 Guido van Rossum 待在家里，发现自己无所事事。于是为了打发“无聊”的时光，他编写了第一版的 Python。

Python 的使用范围非常广，根据开源社区 GitHub 的统计（见图 2-3），在近 10 年它一直是最流行的编程语言之一，比传统的 C、C++语言以及 Windows 系统下十分常用的 C#都更为流行。

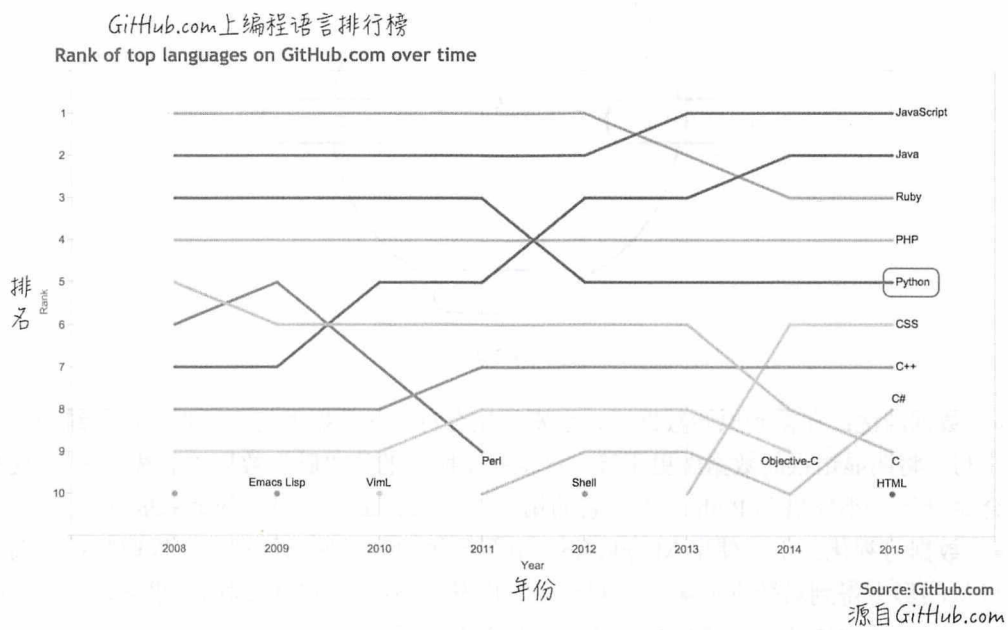


图 2-3

[2] 引自维基百科。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

小安在使用 Python 一段时间后，觉得它是一个专门为非专业程序员设计的编程语言。

- 它的语法结构十分简洁，鼓励大家尽量写容易看明白的代码，尽量写少的代码。
- 从功能方面来说，Python 有大量的标准库和第三方库。小安在这些已有程序的基础上开发自己的应用，能事半功倍，加快开发进度。
- 更方便的是，Python 可以跨平台运行。比如小安常在自己熟悉的 Windows 系统下面编写 Python 代码，然后将开发好的程序部署到 Linux 系统的服务器上。

总结起来就一句话：好学而且好用。

2.1.2 Python 在数据科学中的地位

在掌握 Python 这门编程语言后，小安可以做很多有趣的事情：比如编写网络爬虫程序、从互联网上收集所需要的数据、开发任务调度系统、定时更新模型等。当然作为数据科学工作者的小安，她最常用 Python 做下面这 4 件事情，如图 2-4 所示。

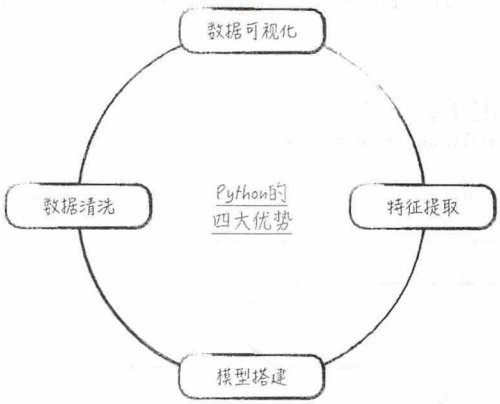


图 2-4

- 数据清洗：在得到原始数据后，小安首先会对这些数据做初步处理，比如统一字符串的大小写、将明显错误的数据做更正等。这也就是所谓的将“脏”数据“清洗干净”，使数据更适合做分析。小安借助 Python 以及它的第三方库 pandas，能很方便地完成这一步工作。
- 数据可视化：小安使用 Matplotlib，用图形化的方式展示数据。在提取特征之前，小安可以从图形中得到对数据的第一直观感受，启发思路；在与其他部门同事交流时，借助图形可以清晰有效地传达与沟通信息，让洞察见解跃然纸上。
- 特征提取：在这一步中，小安通常会先将存放在不同地方的相关数据关联起来，比如将客户基本信息和客户购物信息通过客户 ID 整合到一起。然后对数据做转换，提取出对建模有用的变量，这些变量被称为特征。在这一过程中，小安会用到 Python 的 NumPy、SciPy、

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

pandas 和 PySpark。

- 模型搭建：开源库 scikit-learn、Statsmodels、Spark ML 和 TensorFlow 几乎涵盖了所有常用的基础算法。小安在这些算法库的基础上，根据数据特征和算法假设，可以很容易地将基础算法搭建在一起，造出自己想要的模型。

上述的 4 件事情也正是数据科学中最核心的 4 个步骤。这就难怪小安同其他大部分数据科学家一样，会选择 Python 为工具来完成自己的工作。

2.1.3 不可能绕过的第三方库

表 2-1 列举了在数据科学中最常用的 Python 库。

表 2-1

数据预处理	NumPy	科学计算基础库。它提供高效的 *N* 维数组和向量运算
	SciPy	科学计算库。它依赖与 NumPy，提供高效的数值计算，以及用于函数最优化、数值积分等任务的模块
	pandas	数据结构和数据分析库。包含高级数据结构和类 SQL 语句，让数据处理变得快速、简单
数据可视化	Matplotlib	数据可视化库。它提供大量专业数据图形制作工具
标准模型库	scikit-learn	标准机器学习库。它主要用于分类、回归和聚合等，依赖于 NumPy、SciPy、Matplotlib
	Statsmodels	标准统计模型库。它主要用于假设检验和参数置信区间分析
	Spark ML	分布式机器学习算法库。它可在分布式集群上，如 Hadoop，对大量数据建模。Spark ML 由 Scala 开发，但提供 Python API
	TensorFlow	成熟的深度学习算法库。它提供 GPU 运算模块

后面的章节会陆续用到这些库，读者到时就可以更加直观地感受它们的“威力”。

2.2 Python 安装

介绍了这么多 Python 的好处，那我们赶紧安装它，亲自感受一下吧！

Python 有两个主要的版本：Python 2 和 Python 3。Python 3 是较高的版本，具有 Python 2 所没有的新特性。但由于 Python 3 在设计的时候没有考虑向下兼容，导致实际生产中仍以 Python 2 为主（虽然在本书编写时，Python 3 已经发布快 10 年了）。因此这里推荐读者全新安装时仍使用 Python 2，本书配套的代码也是基于 Python 2（Python 2.7）的。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

下面介绍如何安装 Python 和 2.1.3 节中所列举的库。请读者按照自己使用的计算机系统，参考对应的安装指南。

需要说明的是，分布式机器学习库 Spark ML 涉及 Java 和 Scala 的安装，这里就暂不做介绍。具体的安装细节请见第 11 章。

2.2.1 Windows 下的安装

作者并不推荐大家在 Windows 系统下做开发。原因有很多，其中最重要的一条是：在大数据时代，如前面小安提到，数据都存放在 Linux 系统下。因此，在生产上，数据科学家开发的程序最终将运行在 Linux 环境下。而 Windows 和 Linux 的兼容并不好，很容易导致在 Windows 下开发调试好的程序，在实际生产环境下没办法正常运行。

如果读者使用的计算机是 Windows 系统，可以选择安装 Linux 的虚拟机，然后在虚拟机上做开发。如果读者坚持使用 Windows，由于 TensorFlow 在 Windows 下的限制，只能选择安装 Python 3.5（截止本书编写时）。因此本小节下面的教程也有别于其他章节，使用的是 Python 3。

在 Windows 环境下，最方便的方法就是安装第三方发行版 Anaconda。它将 Python 和许多常用的库打包，包括本书将涉及的 NumPy、SciPy、Matplotlib、scikit-learn 和 Statsmodels，方便读者直接使用。

1. Anaconda 的安装步骤

（1）从网上下载 Anaconda[3]，如图 2-5 所示。

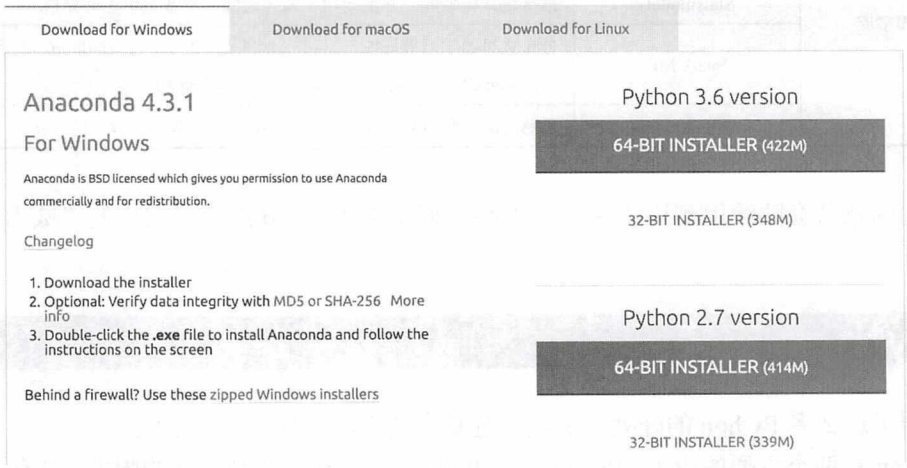


图 2-5

[3] 具体网址和步骤请参考随书配套的代码/ch02-python/python 安装补充说明.html。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

（2）双击下载后的文件，按默认选项安装。如图 2-6 所示。

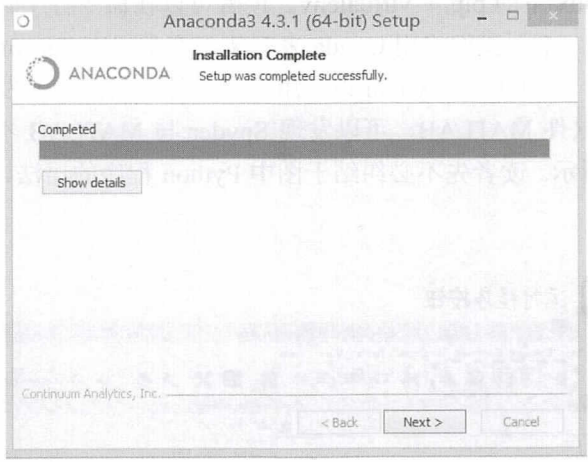


图 2-6

2. 验证安装是否成功

（1）打开“开始”→“应用”。

（2）在搜索栏中输入“cmd”，选择并打开“命令提示符”，如图 2-7 所示。

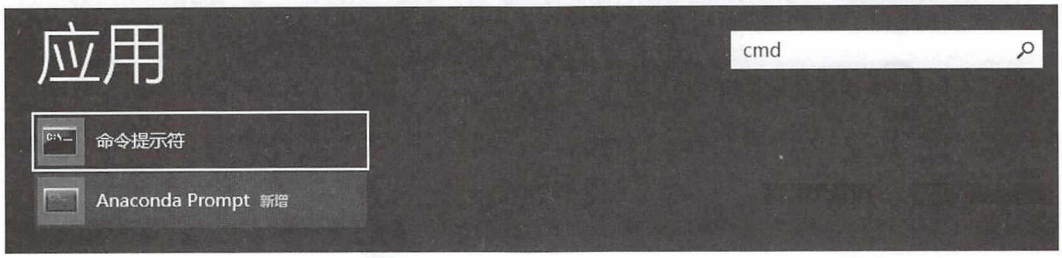


图 2-7

（3）输入“python”并按回车键，如程序清单 2-1 所示。

程序清单 2-1 Python shell on Windows

```
C:\Users\you> python
Python 3.6.0|Anaconda 4.3.1 (64-bit)| (default, Dec 23 2016, 11:57:41) [MSC v.
1900 64 bit (AMD64) on win 32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought you to by Continuum Analytics.
>>> [输入“exit()”退出 Python 返回命令行]
```

Anaconda 在 Windows 下安装了好几个应用程序，如 IPython、Jupyter、Conda 和 Spyder 等。限于篇幅，这里只重点介绍其中的两个——Conda 和 Spyder。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

- Conda：它是一个 Python 开发环境和开源库的管理系统。如果读者对 Linux 比较熟悉，Conda 相当于 Linux 下的 pip + virtualenv。读者可以在命令行中输入“conda list”来列举已安装的 Python 库。下面我们将用 Conda 来安装深度学习库 TensorFlow。
- Spyder：它是一个专门为 Python 设计用于科学计算的集成开发环境（IDE）。如果读者比较熟悉数学分析软件 MATLAB，可以发现 Spyder 与 MATLAB 不论是语法还是界面都非常相似，如图 2-8 所示。读者先不必纠结于图中 Python 程序的语法，我们会在本章的后续内容中介绍它。

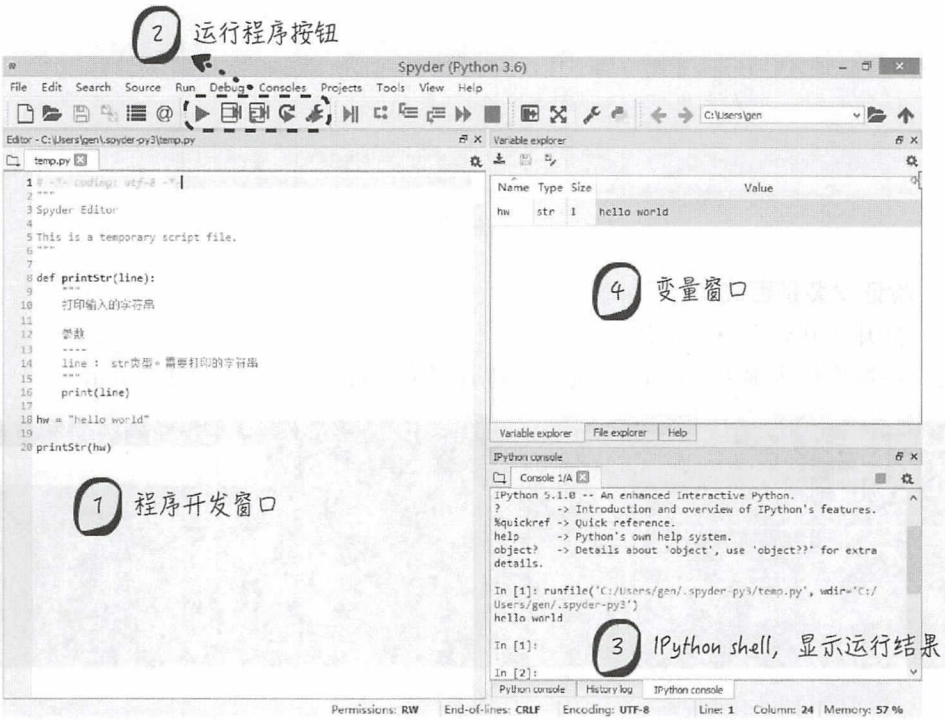


图 2-8

3. 安装 TensorFlow

（1）如上面所述，打开“命令提示符”。

（2）使用如下命令，安装 Python 3.5。

程序清单 2-2 在 Windows 下使用 Conda 安装 Python 3.5

```
C:\Users\you> conda install python=3.5
C:\Users\you> python --version
Python 3.5.3 :: Anaconda custom (64-bit)
```

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

（3）输入如下命令来创建一个 TensorFlow 的环境，并启动创建好的 TensorFlow 环境。

程序清单 2-3　在 Windows 下安装 TensorFlow

```
C:\Users\you> conda create -n tensorflow
C:\Users\you> activate tensorflow
(tensorflow) C:\Users\you> [你的命令行将变成这样，以(tensorflow)开头]
```

（4）输入如下命令安装 TensorFlow。TensorFlow 分为 CPU 版本和 GPU 版本，这里推荐读者，特别是对 GPU 计算不太熟悉的读者安装 CPU 版本。

程序清单 2-3　在 Windows 下安装 TensorFlow

```
[安装 CPU 计算框架的 TensorFlow]
(tensorflow) C:\Users\you> pip install --ignore-installed --upgrade
https://mirrors.tuna.tsinghua.edu.cn/tensorflow/windows/cpu/tensorflow-1.0.1-cp
35-cp35m-win_amd64.whl
[或者安装 GPU 计算框架的 TensorFlow]
(tensorflow) C:\Users\you> pip install --ignore-installed --upgrade
https://mirrors.tuna.tsinghua.edu.cn/tensorflow/windows/gpu/tensorflow_gpu-1.0.
1-cp35-cp35m-win_amd64.whl
```

（5）验证 TensorFlow 安装是否成功。

程序清单 2-3　在 Windows 下安装 TensorFlow

```
C:\Users\you> python
>>> import tensorflow as tf
>>> tf.__version__
'1.0.1'
```

2.2.2　Mac 下的安装

如上面 Windows 下的安装指南所示，可以选择安装第三方发行版 Anaconda。具体的安装步骤这里就不展开了。请读者参考 Windows 下的说明和随书配套的代码。

如同 Anaconda 的 Windows 版本，Anaconda 的 Mac 版本并不包含深度学习库 TensorFlow，需要使用 pip（Python 软件包管理系统）来安装它。虽然使用 pip 需要用到命令行，但操作起来十分简单，甚至比安装 Anaconda 更容易。而且 pip 的应用更广泛，所以建议读者从一开始就尝试用 pip 来安装所需要的库。下面介绍不使用 Anaconda 的安装方法。

从 Mac OS X 10.2 版本开始，Mac 预装了 Python。如果是以学习为目的，可以选择直接使用预装版本的 Python；如果是以开发为目的，预装的 Python 在安装第三方库时容易遇到问题，需要重新安装最新版本的 Python。这里推荐读者重新安装 Python。

1．安装最新版本的 Python

（1）打开“应用程序”→“实用工具”→“终端”，如图 2-9 所示。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

图 2-9

（2）安装 Mac OS 缺失软件包管理器 Homebrew。

程序清单 2-4 在 Mac 上安装 Python

```
1 | [localhost:~] you$ /usr/bin/ruby -e "$(curl -fsSL
https://raw.githubusercon tent.com/Homebrew/install/master/install)"
2 | [需要输入你的电脑密码]
3 | [localhost:~] you$ brew --version
4 | [Homebrew 的版本信息]
```

（3）安装最新版本的 Python 2。

程序清单 2-4 在 Mac 上安装 Python

```
5 | [下面的命令将同时安装最新版本的 Python2 和 pip]
6 | [localhost:~] you$ brew install python
7 | [同时按下 Command 和 N 打开一个新的终端窗口]
8 | [localhost:~] you$ python
9 | Python 2.7.13 (default, Jul 30 2016, 23:25:09)
10 | [GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
11 | Type "help", "copyright", "credits" or "license" for more information.
12 | >>> [同时按下 Control 和 D 退出 Python 返回终端]
13 | [localhost:~] you$ pip --version
14 | [pip 版本以及安装地址]
```

2. 如果需要，单独安装 pip

（1）打开“终端”。

（2）输入并执行如下命令。

程序清单 2-5 在 Mac 上安装 pip

```
1 | [localhost:~] you$ sudo easy_install pip
2 | [需要输入你的电脑密码]
3 | [localhost:~] you$ pip --version
4 | [pip 版本以及安装地址]
```

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

3. 安装需要的库

如果使用 Mac 预装的 Python，可能会遇到报错。

（1）打开“终端”，安装 NumPy。

程序清单 2-6 在 Mac 上安装 Python 库

```
[localhost:~] you$ pip install numpy
[如果你在安装过程中遇到权限问题，使用下面的命令安装]
[localhost:~] you$ sudo -H pip install numpy
[你可以指定 NumPy 安装的版本，比如下面的命令安装 1.11.0 版本的 NumPy]
[localhost:~] you$ pip install numpy==1.11.0
[你还可以使用如下命令更新 NumPy 到最新的版本]
[localhost:~] you$ pip install --upgrade numpy
```

（2）安装 SciPy、pandas、Matplotlib 和 scikit-learn。

程序清单 2-6 在 Mac 上安装 Python 库

```
[localhost:~] you$ pip install pandas
[localhost:~] you$ pip install matplotlib
[localhost:~] you$ pip install pandas
[localhost:~] you$ pip install scikit-learn
[如果你遇到权限问题，或者想升级库的版本，请参考第一步 Numpy 的安装指南]
```

（3）安装 Statsmodels。

程序清单 2-6 在 Mac 上安装 Python 库

```
[localhost:~] you$ pip install --upgrade patsy
[localhost:~] you$ pip install statsmodels
```

（4）安装 TensorFlow。推荐读者安装 CPU 版本的 TensorFlow。

程序清单 2-6 在 Mac 上安装 Python 库

```
[因为依赖的原理，使用如下命令卸载已安装的 Tensorflow]
[localhost:~] you$ sudo pip uninstall tensorflow
[安装 CPU 计算框架的 TensorFlow]
[localhost:~] you$ pip install tensorflow
[或者安装 GPU 计算框架的 TensorFlow]
[localhost:~] you$ pip install tensorflow-gpu
```

（5）验证安装是否成功。

程序清单 2-6 在 Mac 上安装 Python 库

```
[localhost:~] you$ python
>>> import numpy as np
>>> import scipy
>>> import pandas as pd
>>> import sklearn
>>> import matplotlib.pyplot as plt
>>> import statsmodels
>>> import tensorflow as tf
```

书籍是人类进步的阶梯


```
>>> [同时按下 Control 和 D 退出 Python 返回终端]
```

4. 安装 IPython

IPython 是一款基于 Python 的交互式解释器，能大大提高 Python 开发的效率。如果读者不理解交互式解释器这个名词，先不用担心，第 2.3 节将做详细的介绍。安装 IPython 的步骤如下。

程序清单 2-6 在 Mac 上安装 Python 库

```
[localhost:~] you$ pip install ipython
[localhost:~] you $ ipython
Python 2.7.13 (default, Jul 30 2016, 23:25:09)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: exit
[localhost:~] you$
```

2.2.3 Linux 下的安装

同 Mac 相似，Anaconda 也提供 Linux 版本。具体的安装步骤请参考 Windows 下的说明和随书配套的代码。

Linux 的版本很多，但限于篇幅，这里只介绍在 Ubuntu 上的安装。下面的安装指南或许也能在其他版本的 Linux 上运行，但我们只在 Ubuntu 14.04 或者更高的版本上测试过这些安装步骤。

虽然 Ubuntu 有预装的 Python，但版本比较陈旧，推荐安装较新版本的 Python。

1. 安装 Python

（1）打开“搜索”，输入“terminal”，打开终端，如图 2-10 所示。

图 2-10

书籍是人类进步的阶梯

（2）在“终端”里面输入并运行下面的命令。

程序清单 2-7 在 Ubuntu 上安装 Python

```
[更新 apt-get 的下载列表]
you@you:~$ sudo apt-get install software-properties-common
[需要输入你的电脑密码]
you@you:~$ sudo add-apt-repository ppa:fkrull/deadsnakes-python2.7
you@you:~$ sudo apt-get update
[使用 apt-get 下载最新版的 Python2]
you@you:~$ sudo apt-get install python2.7
you@you:~$ python --version
[安装的 Python 版本]
```

2. 安装 pip

pip 是 Python 软件包管理系统，方便我们安装所需的第三方库。安装 pip 的步骤如下。

（1）打开“终端”。

（2）输入并运行如下代码。

程序清单 2-8 在 Ubuntu 上安装 pip

```
[使用 apt-get 安装 pip 以及相应的依赖]
you@you:~$ sudo apt-get install python-pip
[升级 pip 到较新的版本]
you@you:~$ sudo -H pip install --upgrade pip==9.0.1
you@you:~$ pip --version
'pip 9.0.1 from /usr/local/lib/python2.7/dist-packages (python 2.7)'
```

3. 安装所需要的库

（1）打开“终端”，安装 NumPy、SciPy、Matplotlib、pandas 和 Statsmodels。

程序清单 2-9 在 Ubuntu 上安装 Python 库

```
you@you:~$ sudo apt-get install python-numpy
you@you:~$ sudo apt-get install python-scipy
you@you:~$ sudo apt-get install python-matplotlib
[下面这个命令将同时安装 pandas 和 Statsmodels]
you@you:~$ sudo apt-get install python-pandas
```

（2）上面所安装的 Python 库版本比较陈旧，请升级版本。

程序清单 2-9 在 Ubuntu 上安装 Python 库

```
you@you:~$ sudo -H pip install --upgrade numpy scipy matplotlib pandas statsmodels
```

（3）安装 scikit-learn 和 TensorFlow。推荐读者安装 CPU 版本的 TensorFlow。

程序清单 2-9 在 Ubuntu 上安装 Python 库

```
[预防编码错误]
you@you:~$ export LC_ALL=C
[安装 scikit-learn]
```

书籍是人类进步的阶梯


```
you@you:~$ sudo -H pip install scikit-learn
[安装 CPU 计算框架的 TensorFlow]
you@you:~$ sudo apt-get install python-dev
you@you:~$ sudo -H pip install tensorflow
[或者安装 GPU 计算框架的 TensorFlow]
you@you:~$ sudo -H pip install tensorflow-gpu
```

（4）验证安装是否成功。

程序清单 2-9 在 Ubuntu 上安装 Python 库

```
you@you:~$ python
>>> import numpy as np
>>> import scipy
>>> import pandas as pd
>>> import sklearn
>>> import matplotlib.pyplot as plt
>>> import statsmodels
>>> import tensorflow as tf
>>> [输入 exit 退出]
```

4. 安装 IPython

如同在 Mac 系统上一样，安装 IPython，具体说明请参考 2.2.2 节。

程序清单 2-9 在 Ubuntu 上安装 Python 库

```
you@you:~$ sudo -H pip install ipython
you@you:~$ ipython
Python 2.7.12 (default, Jul 18 2016, 23:25:09)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: exit
you@you:~$
```

2.3 Python 上手实践

现在，我们已经安装好 Python 了，是不是已经有点迫不及待想要试试它？下面让我们开始动手实践吧。

2.3.1 Python shell

Python 作为一款动态语言，通常有两种使用方法：可以把它当脚本解释器，运行已编辑

书籍是人类进步的阶梯

好的程序脚本；同时 Python 提供一个实时交互的命令窗口（Python shell），可以在其中输入并运行任何 Python 语句，如图 2-11 所示。这让学习、调试和测试 Python 语句变得十分容易。

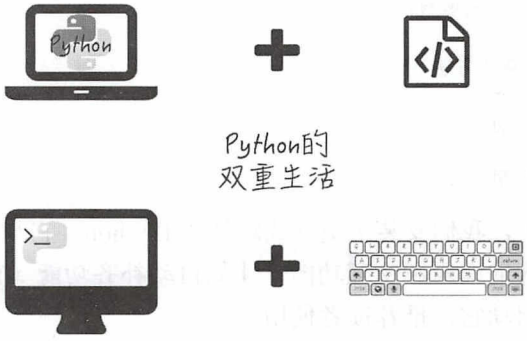


图 2-11

如程序清单 2-10 所示，在终端（Linux 或 Mac）或者命令提示符（Windows）里输入“python”启动 Python shell。

（1）可以在 Python shell 里面对变量赋值，然后对使用变量进行计算。而且只要不关闭 shell，就可以一直使用这些变量。如第 1～3 行代码所示。值得注意的是，Python 是所谓的动态类型语言，所以在变量赋值的时候不需要声明变量的类型。

（2）Python shell 里面可以运行任意的 Python 语句，如第 5 行代码所示，因此甚至有人把它当作计算器。

（3）也可以在 shell 中导入并使用第三方库，如第 7、8 行代码所示。需要注意的是，如第 7 行代码所示，在导入第三方库“numpy”的同时可以给它取一个别名，如“np”。在后面需要使用“numpy”时，就用“np”代替，减少字符输入量。

（4）下面介绍 3 个特别实用的小技巧。

- 使用 type 函数得到调用对象的类型。
- 使用 dir 函数得到调用对象的所有属性和方法，如第 10、12 行代码所示。
- Python 里的每一个对象（类、函数）都有一个默认的“__doc__”变量（值得注意的是，“doc”前后是双下划线），里面记录了对象的使用说明，如第 14 行代码所示。

当对所使用的对象不是很熟悉时，这 3 个方法可以帮助我们迅速地了解它们。

程序清单 2-10 Python shell

```
1 |  >>> a = 1
2 |  >>> b = 2
3 |  >>> print a + b
4 |  3
5 |  >>> 10 / 2.0
6 |  5
```

书籍是人类进步的阶梯


```
>>> import numpy as np
>>> np.max([1, 2, 3])
3
>>> type(a)
<type 'int'>[a 的类型]
>>> dir(np)
[np 所有的类变量和方法]
>>> print np.max.__doc__
[np.max 方法说明]
>>> help(np.max)
[np.max 方法说明，按 q 键退出]
```

在之前的安装步骤中，我们安装了交互式解释器 IPython。它与上面介绍的 Python shell 相似，但提供了更为强大的编辑和交互功能，比如自动补齐功能（Tab 键）。类似地，在终端输入“ipython”就能启动它，推荐读者使用。

2.3.2 第一个 Python 程序：Word Count

在以往的 IT 类书籍中，介绍一种编程语言的第一个程序都是“Hello World”。但本书针对数据科学，所以第一个 Python 程序也与时俱进地改为了大数据版的“Hello World”：“Word Count”。完整的程序脚本放在随书配套的代码/ch02-python/word_count.py。在终端里执行“python word_count.py”命令就可以运行这个程序了，下面让我们来仔细看看这个程序的代码。

程序清单 2-11 Word Count

```
# -*- coding: UTF-8 -*-

def wordCount(data):
    """
    输入一个字符串列表，统计列表中字符串出现的次数

    参数
    ----
    data : list[str], 需要统计的字符串列表

    返回
    ----
    re : dict, 结果 hash 表，key 为字符串，value 为对应的出现次数
    """
    re = {}
    for i in data:
        re[i] = re.get(i, 0) + 1
    return re


if __name__ == "__main__":
    data = ["ab", "cd", "ab", "d", "d"]
    print "The result is %s" % wordCount(data)
    # The result is {'ab': 2, 'd': 2, 'cd': 1}
```

书籍是人类进步的阶梯

（1）程序脚本的第一行表示这个脚本的编码为 UTF-8。如果脚本需要用到中文字符，不要忘记在脚本开头加上这一句。

（2）程序清单 2-11 的第 3～18 行代码在定义 wordCount 函数。这个函数有一个参数“data”。正如前面介绍的，读者可以看到 Python 不使用大括号，而使用缩进来定义语句块，如图 2-12 所示。因此在书写 Python 代码时，需要特别关注代码的对齐。这是为什么在程序员中流传着开发 Python 代码需要购买游标卡尺的说法。当然这只是个玩笑话，常见的开发工具如 Vim、IPython、Spyder 等都提供代码自动对齐功能。

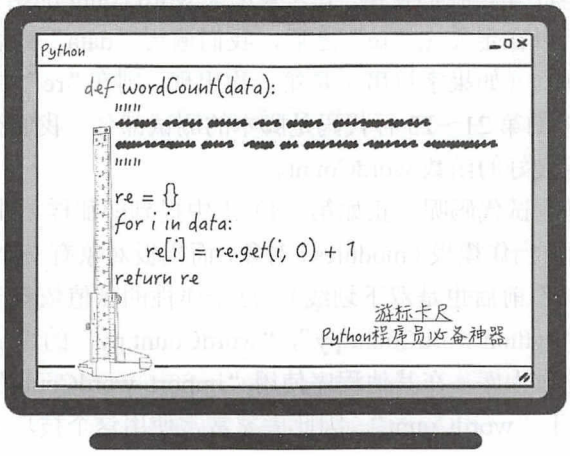


图 2-12

（3）函数名的下面是一个字符串，如第 4～14 行代码所示。它是函数的说明文档，记录着函数功能的描述，以及函数参数和返回值的定义。这个字符串虽然对函数功能没有任何影响，甚至可以不写，但它对 Python 程序工程化非常重要。假设你和小安是同事，在某个项目中，你需要调用小安开发的函数 *A*。而小安在开发时为图简单，并没有给函数 *A* 写上说明文档。由于 Python 的语法规定一个函数在定义时，既不需要声明参数类型，也不需要定义返回值的个数和它们的类型，无法做静态类型检查。你沮丧地发现，即使通读一遍小安的源代码，也很难知晓如何使用这个函数，甚至不知道应该传送怎样的参数，会得到什么样的返回。于是，你要求小安按照如上面的格式补充说明文档。小安按要求修改后，你就可以在使用时，通过命令“A.__doc__”来阅读对应的说明文档（参考 2.3.1 节）。

（4）如果读者以前接触过其他编程语言（比如 Java），会觉得奇怪：*A* 不是一个函数么，为什么函数也有属性“__doc__”呢？这是 Python 的另外一个特性。在 Python 中，所有东西都是对象，连函数也是对象，所以 *A* 作为函数也有自己的属性。

（5）在函数说明文档下面，我们定义了一个空的 dict 类型变量“re”。dict 类型可以翻译

书籍是人类进步的阶梯

为字典类型，是 Python 的 5 个基本类型之一。现实中的字典里面记录了字和对应的解释。从结构上看，字典其实是一堆键值对（key value pair），字为键（key），而对应的解释为值（value）。而 dict 也一样，是一个键值对的集合。如果读者对数据结构比较了解，会发现 dict 其实就是一个 Hash 表。如果使用 Python 开发，就会发现 dict 十分灵活、实用。它也是作者最喜欢的数据类型。

（6）在 Python 中，dict 的书写形式如下：用“{}”将所有值括起来，健与值之间用“:”隔空，键值对之间用“,”隔开，如第 24 行代码所示。

（7）在程序清单 2-11 里，我们使用“re”来记录 word count 的结果，即键为字符串，值为对应的出现次数。所以在定义完“re”之后，我们遍历“data”列表中的每一个字符串，并对“re”中对应的值加 1（如果字符串 x 是第一次出现，则在“re”中插入键值对（x: 1））。

（8）程序清单 2-11 的第 21～23 行代码是脚本的测试部分。我们在这里创建了一个字符串列表，并将它传给定义好的函数 wordCount。

（9）为什么说它是测试代码呢？正如第（4）点中提到的那样，在 Python 中，所有内容都是对象。脚本本身也被当作模板（modules）对象，而模板对象有一个内置属性“__name__”（值得注意的是，“name”前后也是双下划线）。这个属性的赋值依赖于用户怎么使用它：如果直接运行它，比如“python wordCount.py”，“wordCount.py”的“__name__”属性就等于“__main__”。如果把它作为库，在其他程序使用“import wordCount”时，wordCount.py 的“__name__”属性就等于“wordCount”。因此大家常常使用这个技巧来做简单的测试，或者把它作为程序的主入口，比如 Java 中的 main 函数。

读到这里，希望读者已经能理解 word_count.py 中的 Python 代码了。下面将讨论更多 Python 编程的基础知识。

2.3.3 Python 编程基础

本小节将介绍 Python 的 3 个常用的基本数据类型：dict、list 和 tuple，Python 特色的 lambda 表达式以及内置函数 map、reduce 和 filter。

1. Python 的基本数据类型

（1）dict 类型在上一节已经见过。下面是它的基本操作代码展示。最好的学习方法就是动手实践，所以鼓励读者打开 Python shell，跟着本书一起练习。

程序清单 2-12 Python 基本数据类型——dict

```
1 | # dict 基本操作
2 | ## 初始化一个 dict 类型变量 x
3 | >>> x = {"a": 1, "b": 2, "c": 3}
4 | >>> print x
5 | {'a': 1, 'c': 3, 'b': 2}
```

书籍是人类进步的阶梯


```
6  |  >>>
7  |  ## 读取 x 中的一个元素，直接读取或者使用"get 方法
8  |  >>> print x["a"], ",", x.get("a")
9  |  1 , 1
10 |  ## 读取 x 中不存在的一个元素，注意直接读取会报错
11 |  >>> print x["d"]
12 |  Traceback (most recent call last):
13 |    File "<stdin>", line 2, in <module>
14 |  KeyError: 'd'
15 |  ## 使用"get 方法返回 None 值；可以在"get 方法中使用默认值
16 |  >>> print x.get("d")
17 |  None
18 |  >>> print x.get("d", "No such key")
19 |  No such key
20 |  >>>
21 |  ## 修改 dict
22 |  >>> x["c"] = 4
23 |  ## 插入新的键值对
24 |  >>> x["d"] = "5"
25 |  >>> print x
26 |  {'a': 1, 'c': 4, 'b': 2, 'd': '5'}
27 |  ## 删除键值对
28 |  >>> del x["c"]
29 |  >>> print x
30 |  {'a': 1, 'b': 2, 'd': '5'}
```

（2）list 就是列表类型，可以将任意元素放进一个列表里。

程序清单 2-13 Python 基本数据类型——list

```
1  |  # list 基本操作
2  |  ## 初始化一个 list 类型变量 y
3  |  >>> y = ["A", "B", "C", "a", "b", "c"]
4  |  >>> print y
5  |  ['A', 'B', 'C', 'a', 'b', 'c']
6  |  >>>
7  |  ## 读取 y 中的元素
8  |  >>> print y[0]
9  |  A
10 |  >>> print y[-1]
11 |  c
12 |  >>> print y[0: 3]
13 |  ['A', 'B', 'C']
14 |  ## 查找 y 中的元素
15 |  >>> print y.index("a")
16 |  3
17 |  >>>
18 |  ## 修改 list
19 |  >>> y[0] = 4
20 |  >>> print y
21 |  [4, 'B', 'C', 'a', 'b', 'c']
22 |  ## 在 y 的最后面插入新元素
23 |  >>> y.append(5)
```

书籍是人类进步的阶梯


```
24 | >>> print y
25 | [4, 'B', 'C', 'a', 'b', 'c', 5]
26 | ## 在指定位置插入新元素
27 | >>> y.insert(3, "new")
28 | >>> print y
29 | [4, 'B', 'C', 'new', 'a', 'b', 'c', 5]
30 | ## 两个 list 合并，注意和 append 的区别
31 | >>> print y + ["d", "e"]
32 | [4, 'B', 'C', 'new', 'a', 'b', 'c', 5, 'd', 'e']
33 | >>> y.append(["d", "e"])
34 | >>> print y
35 | [4, 'B', 'C', 'new', 'a', 'b', 'c', 5, ['d', 'e']]
36 | ## 删除 y 里面的元素
37 | >>> y.remove("B")
38 | >>> print y
39 | [4, 'C', 'new', 'a', 'b', 'c', 5, ['d', 'e']]
```

（3）tuple 也称为元祖类型，与列表功能非常相似，不同之处是元祖里面的元素不能被修改。

程序清单 2-14 Python 基本数据类型——tuple

```
1  | # tuple 基本操作
2  | ## 初始化一个 tuple 类型变量 z
3  | >>> z = ("a", "b", "c", "d", "e")
4  | >>> print z
5  | ('a', 'b', 'c', 'd', 'e')
6  | >>>
7  | ## 读取 z 中的元素，与 list 类似
8  | >>> print z[0]
9  | a
10 | >>> print z[-1]
11 | e
12 | >>> print z[0: 3]
13 | ('a', 'b', 'c')
```

2. lambda 表达式和内置函数 map、reduce、filter

在 Python 中，lambda 表达式类似于函数。它以关键字“lambda”开头，之后就是函数的参数列表，并以“:”结尾。紧接着就是表达式主体，整个只是一行语句。这样说可能很抽象，我们来看下面这个具体的例子。比如下面的 f 和 g 是等价的。f 是普通定义的函数，而 g 是 lambda 表达式。

程序清单 2-15 lambda 表达式和常用内置函数

```
1 | >>> def f(a, b):
2 | ...     return a + b
3 | ...
4 | >>> print f(1, 2)
5 | 3
6 | >>> g = lambda x, y: x + y
7 | >>> print g(1, 2)
8 | 3
```

书籍是人类进步的阶梯

有的读者可能会有疑问，既然 g 与函数 f 等价，那什么时候会用到 lambda 表达式呢？答案是创建匿名函数时。我们先看下面这个例子：

程序清单 2-15 lambda 表达式和常用内置函数

```
 9 | ## 内置 map 函数和 lambda 表达式
10 | ## l 是一个 0～5 的列表
11 | >>> l = range(6)
12 | >>> print l
13 | [0, 1, 2, 3, 4, 5]
14 | ## 下面的操作将生成一个新的列表，列表里面的元素为 l 里元素加 1
15 | >>> def addOne(data):
16 | ...     re = []
17 | ...     for i in data:
18 | ...         re.append(i + 1)
19 | ...     return re
20 | ...
21 | >>>
22 | >>> print addOne(l)
23 | [1, 2, 3, 4, 5, 6]
24 | ## 通过内置的 map 函数和 lambda 表达式可以达到同样的效果
25 | >>> print map(lambda x: x + 1, l)
26 | [1, 2, 3, 4, 5, 6]
27 | ## 达到同样功能的列表生成式
28 | >>> print [i + 1 for i in l]
29 | [1, 2, 3, 4, 5, 6]
```

内置的 map 函数有两个参数，第一个是一个函数；第二个是一个可遍历的对象，比如 list、tuple 和 dict 等。map 函数将第二个参数里的每个元素依次传给作为第一个参数的函数，并将函数的返回值依顺序组成一个列表。

如第 25 行代码所示，map 函数将“l”里面的每一个数字按顺序传给 lambda 表达式，得到加 1 后的值。并将这些值依次插入结果列表里面。计算过程和上面第 15～19 行代码定义的 addOne 函数一样。

在第 25 行代码中，lambda 表达式就定义了一个匿名函数，因为它起到了函数的功能，但又没有任何标识符来代表它，所以很形象地被命令为匿名函数。

上面的第 28 行代码是一种很简洁的书写方式，被称为“列表生成式”。在实际的 Python 开发中，应用也特别广泛。对于第一次接触 Python 的读者来说，这样的书写方式显得怪怪的。不过慢慢地，你就会发现它的好处并爱上它。比如下面的这个例子。

程序清单 2-15 lambda 表达式和常用内置函数

```
30 | ## 计算 l 中每个元素的两倍和平方，并将两种组成一个列表
31 | ## lambda 表达式和 Python 函数一样，也可以接受函数作为参数
32 | >>> twoTimes = lambda x: x * 2
33 | >>> square = lambda x: x ** 2
34 | >>> print [map(lambda x: x(i), [twoTimes, square]) for i in l]
35 | [[0, 0], [2, 1], [4, 4], [6, 9], [8, 16], [10, 25]]
```

书籍是人类进步的阶梯

一行 Python 语句就搞定了，是不是很方便呢？最后，我们来看看 filter 和 reduce 的用法。

程序清单 2-15 lambda 表达式和常用内置函数

```
## 内置 filter 函数，选择 l 中的偶数
>>> filter(lambda x: x % 2 == 0, l)
[0, 2, 4]
## 内置 reduce 函数，计算 l 的和
>>> reduce(lambda accumValue, newValue: accumValue + newValue, l, 0)
15
```

filter 函数和 map 函数非常相似。只不过 filter 函数是用来做筛选的。它会根据作为第 1 个参数的函数返回值来剔除掉返回值为 0 的元素。而 reduce 函数是用来做聚合运算的。以上面的代码为例，第 1 个参数是相加函数，第 2 个参数是需要求和的列表“1”，第 3 个（可选）参数为初始值 0。

以上就是 Python 编程的基础知识。

2.3.4 Python 的工程结构

本节将讨论 Python 项目的工程结构。这部分内容对初学者来讲，可能比较抽象。如果读者发现下面的文字生涩难懂，可先跳过此节，这并不影响本书其他章节的阅读。

假设小安已经开发好了一些脚本，并把它们放在某个文件目录下。现在，作为读者的你在开发新项目的时候，希望能像使用 NumPy 这种第三方库一样复用之前的代码。这应该如何实现呢？答案是创建充满“魔力”的__init__.py 文件。具体来看下面这个例子[4]。

首先创建一个 mini_project 的目录，下面有两个子目录：components 和 tests。而 components 下面有两个 Python 脚本，即 counter.py 和 selecter.py，如图 2-13a 所示。

（1）在 counter.py 脚本中定义了 wordCount 函数。

（2）在 selecter.py 中定义了 getFrequentItem 函数。这个函数依赖于上面提到的 wordCount 函数，因此在脚本的开头部分使用如下命令来导入它：

```
from mini_project.components.counter import wordCount
```

（3）tests/test_selecter.py 是程序的入口，也就是将被直接运行的脚本，这个脚本会调用 getFrequentItem 函数。类似地，在此脚本开头导入 getFrequentItem。

```
from mini_project.components.selecter import getFrequentItem
```

如果这时使用“python test_selecter.py”命令来运行程序，将会得到如下的错误提示：

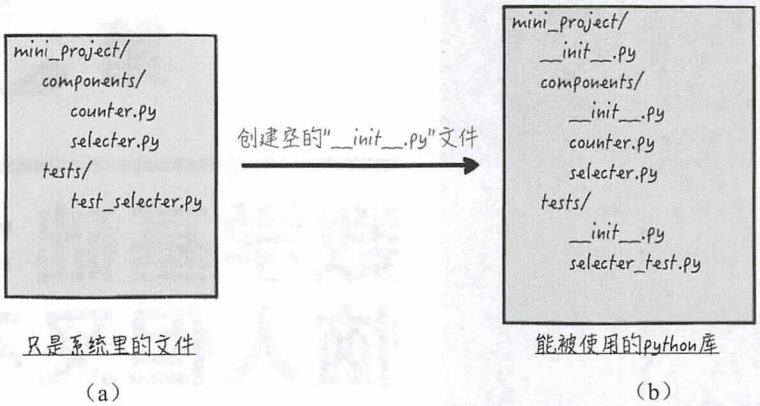
```
ImportError: No module named mini_project.components.selecter
```

这是因为 Python 并没有将目录 mini_project 当成一个可以使用的程序库，所以导入失败。

[4] 完整的实现请参考随书配套的代码/ch02-python/mini_project/。

书籍是人类进步的阶梯

想要修复这个 bug，只需如图 2-13b 一样在各个目录下创建一个空的__init__.py 文件。



只是系统里的文件

（a）

能被使用的python库

（b）

图 2-13

从本质上讲，Python 的库其实就是一个包含__init__.py 文件的目录。__init__.py 定义了这个库的属性和方法。当我们使用 import 命令导入一个库时，其实是在导入__init__.py 这个文件。通常情况下，不需要在它里面定义任何内容，只需一个空文件即可，Python 会自动按默认设置处理。但如果没有这个文件，Python 就不会把对应目录当作第三方库，我们也就没办法导入和使用它了。

另外值得提醒的是，当要导入一个库时，需要确保它对应的目录在系统路径下可见。说得通俗一点就是在“sys.path”下面能找到对应的库目录。如果没有，则需要将相应的路径加到“sys.path”里（读者有没有发现，“sys.path”其实就是一个 list。）。

2.4 本章小结

本章我们介绍了数据科学的常用 IT 工具——Python 和相关的第三方开源库，主要内容包括安装指南、编程基础和 Python 的工程结构。

本章的目的是让大家对 Python 有大致的了解，为后面的章节做准备。不知道你是否已达到这个目的？如果你仍有疑惑，不必担心，在之后的章节里，我们会结合具体例子讲解更多 Python 语言的细节，不断地重复和动手练习。相信读者在读完本书后，能比较熟练地使用 Python。

当然 Python 编程还有很多内容，本书并不是 Python 编程教程，所以这里就不做更多的介绍。有兴趣的读者可以阅读其他 Python 书籍，比如 *Dive into Python*。

书籍是人类进步的阶梯

第3章

数学基础：恼人但又不可或缺的知识

If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is.

（如果有人不相信数学是简单的，那是因为他们没有意识到人生有多复杂。）

——*John von Neumann*

3.1 矩阵和向量空间

3.2 概率：量化随机

3.3 微积分

3.4 本章小结

书籍是人类进步的阶梯

毫无疑问，数学是数据科学的灵魂。不管是机器学习或是统计学模型，从本质上讲都是数学模型，因此，扎实的数学知识对于理解模型假设和分析模型结果至关重要。可能很多读者会觉得“数学”这两个字就代表了枯燥和难以理解，事实也许有时如此，但正如“计算机之父”冯·诺依曼说的那样，我们并不理解数学，只是越来越习惯它[1]。本章的目的也一样，就是让读者熟悉数据科学中常遇到的数学概念和数学符号，在阅读本书后续章节或者参考其他文献时，不再害怕面对它们。本章将简单介绍 3 个方面的数学知识。

- 矩阵和向量空间：矩阵是数据的基本表示形式，本书后面的所有章节都会涉及这部分内容。
- 概率：概率作为量化随机性的工具，是统计学和贝叶斯框架的核心内容。第 4、5、7、9 章中会有大量的概率运算及推导。
- 微积分：它是计算机解决最优化问题（也就是求函数的最值）的理论基础，出现频率较高的章节为第 6、12、13 章。

对数学特别不感兴趣的读者，可先匆匆浏览这部分内容，然后继续阅读其他章节。当遇到不太理解的数学概念时，再回来仔细阅读本章。另外为了读者查找方便，本章将在正文中把涉及的数学概念用加粗的字体表示。

3.1 矩阵和向量空间

在数据科学领域，数据和模型的存在形式都是矩阵，而后者则直观地表示为向量空间里的点。矩阵和向量空间这两个数学概念是本节讨论的内容。

3.1.1 标量、向量与矩阵

首先通过一个简单例子来感性认识一下标量（scalar）、向量（vector）和矩阵（matrix）这 3 个数学概念。

假设我们设计了一款网络对战游戏，在游戏中，玩家选择自己的英雄与其他玩家对战。每个英雄的能力由 3 种属性描述：智力、敏捷和力量[2]。为了方便表示，不妨用i表示智力、a表示敏捷、s表示力量。

对于英雄 A，它的设定是智力型英雄，智力被设定为 10，敏捷为 6 以及力量为 2，用数

[1] 约翰·冯·诺依曼（John von Neumann），美国数学家，在计算机、量子力学和经济学都有突出贡献。他的这句名言的原文为“Young man, in mathematics you don't understand things. You just get used to them.”（年轻人，在数学中你不会理解事情，你只是慢慢习惯它们）。

[2] 如果读者对 DotA 比较熟悉，不妨将这款游戏想象成 DotA。

书籍是人类进步的阶梯

学式子表示为$i = 10, a = 6, s = 2$。换句话说，我们用数字表表示各个属性具体的值，这在数学上就叫作标量，标量其实就是数字。

将这 3 个属性按照智力、敏捷和力量的顺序写在一起，就可以表示一个英雄的能力了。比如用$\boldsymbol{A} = (10, 6, 2)$表示英雄 A。在数学上$\boldsymbol{A}$被称为向量，正确地说应该是行向量。直观上，行向量是多个数字（标量）排成一行。与之类似的是列向量，即多个数字排成一列。

现在我们设计了另外 3 个英雄，分别为 B、C 和 D，向量表示为$\boldsymbol{B} = (3, 4, 10)$、$\boldsymbol{C} = (5, 10, 4)$和$\boldsymbol{D} = (6, 9, 5)$。将这 4 个英雄的向量排列成矩形阵列，即每一行表示一个英雄，得到图 3-1 所示的矩阵，而这个矩阵就可以表示所有 4 个英雄的属性数据。

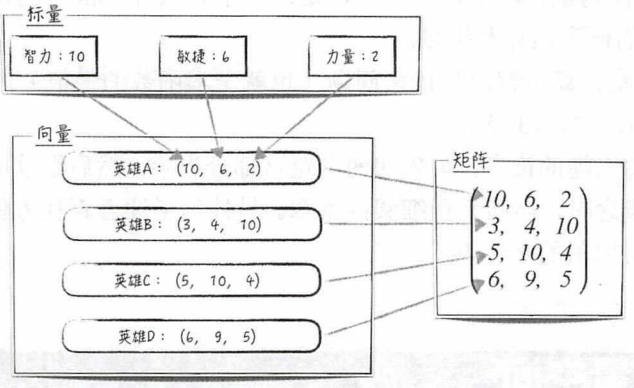


图 3-1

用学术语言来定义矩阵：一个$n \times m$的矩阵，是一个由n行m列元素排列成的矩形阵列。比如公式（3-1）表示的就是一个4×3的矩阵。从数学上来讲，标量和向量其实是比较特殊的矩阵。标量可以被看作一个1×1的矩阵，而包含k个数字行向量（也称为k维行向量）可以看作一个$1 \times k$的矩阵，而包含k个数字的列向量可以被认为是一个$k \times 1$的矩阵。

在数学上，通常如公式（3-1）所示表示向量和矩阵，其中$x_{i,j}$表示标量，也就是一个实数，$\boldsymbol{X}_i$表示一个m维的行向量，$\boldsymbol{X}$表示$n \times m$的矩阵，$\mathbf{R}^{n \times m}$表示所有取值为实数的$n \times m$矩阵全体。本书后面的章节也采用相同的记号。需要注意的是，列向量可以表示为行向量的转置，因此没有专门记号来表示列向量。转置运算的细节请参考 3.1.3 节。

$$\boldsymbol{X}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,m})$$

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{X}_1 \\ \boldsymbol{X}_2 \\ \vdots \\ \boldsymbol{X}_n \end{pmatrix} = (x_{i,j}) \in \mathbf{R}^{n \times m} \tag{3-1}$$

在实际生产中，与上面游戏的例子类似，我们用矩阵来表示搭建模型所用的数据和模型本身的参数。相关的运算，比如估计模型参数或者根据自变量预测结果等，其实都是矩阵运算。

书籍是人类进步的阶梯

3.1.2 特殊矩阵

在讨论矩阵运算之前，先来看一类特殊的矩阵：**方阵**（squared matrix）。它是行数等于列数的矩阵。从形状上来看，它就像一个正方形，因此被称为方阵。有 3 种方阵需要特别注意。

- **单位矩阵**（identity matrix），矩阵的对角线元素等于 1，其他元素等于 0，记为$\boldsymbol{I_n}$。

$$\boldsymbol{I_n} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} = \left(1_{\{i=j\}}\right) \in \mathbf{R}^{n \times n} \tag{3-2}$$

- **对角矩阵**（diagonal matrix），除矩阵的对角线元素外，其他元素都等于 0，记为$diag(d_1, d_2, \ldots, d_n)$。不难注意到，单位矩阵是一种特殊的对角矩阵。

$$diag(d_1, d_2, \ldots, d_n) = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} \in \mathbf{R}^{n \times n} \tag{3-3}$$

- **三角矩阵**（triangular matrix），它可以细分为上三角矩阵和下三角矩阵两种。上三角矩阵的对角线下方的元素全部为零，记为$\boldsymbol{U}$；下三角矩阵的对角线上方的元素全部为零，记为$\boldsymbol{L}$。不难发现，对角矩阵是一种特殊的三角矩阵。

$$\boldsymbol{U} = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & u_{n,n} \end{pmatrix} \in \mathbf{R}^{n \times n}$$

$$\boldsymbol{L} = \begin{pmatrix} l_{1,1} & 0 & \cdots & 0 \\ l_{2,1} & l_{2,2} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ l_{n,1} & l_{n,2} & \cdots & l_{n,n} \end{pmatrix} \in \mathbf{R}^{n \times n} \tag{3-4}$$

3.1.3 矩阵运算

为了能像使用数字一样使用矩阵，我们为它定义了“加减乘除”四种运算。

1. 矩阵的加减法

（1）与数字的加减法不同，并不是任何两个矩阵都可以进行加减运算，要求矩阵的形状是一样的，也就是它们的行数和列数都相等。假设矩阵$\boldsymbol{X}, \boldsymbol{Y}$同为 $n \times m$的矩阵，则它们的和差仍为$n \times m$的矩阵，具体的加减法定义如下：

$$\boldsymbol{X} = \left(x_{i,j}\right) \in \mathbf{R}^{n \times m};\ \boldsymbol{Y} = \left(y_{i,j}\right) \in \mathbf{R}^{n \times m}$$

$$\boldsymbol{X} \pm \boldsymbol{Y} = \begin{pmatrix} x_{1,1} \pm y_{1,1} & \cdots & x_{1,m} \pm y_{1,m} \\ \vdots & & \vdots \\ x_{n,1} \pm y_{n,1} & \cdots & x_{n,m} \pm y_{n,m} \end{pmatrix} = \left(x_{i,j} + y_{i,j}\right) \in \mathbf{R}^{n \times m} \tag{3-5}$$

书籍是人类进步的阶梯

（2）根据上面的定义，不难得出，矩阵的加法也满足结合律和交换律。假设$\boldsymbol{Z}$也是一个$n \times m$的矩阵，可以得到如下的公式：

$$\boldsymbol{X} + \boldsymbol{Y} = \boldsymbol{Y} + \boldsymbol{X}$$
$$\boldsymbol{X} + \boldsymbol{Y} + \boldsymbol{Z} = \boldsymbol{X} + (\boldsymbol{Y} + \boldsymbol{Z}) \tag{3-6}$$

2. 矩阵的乘法

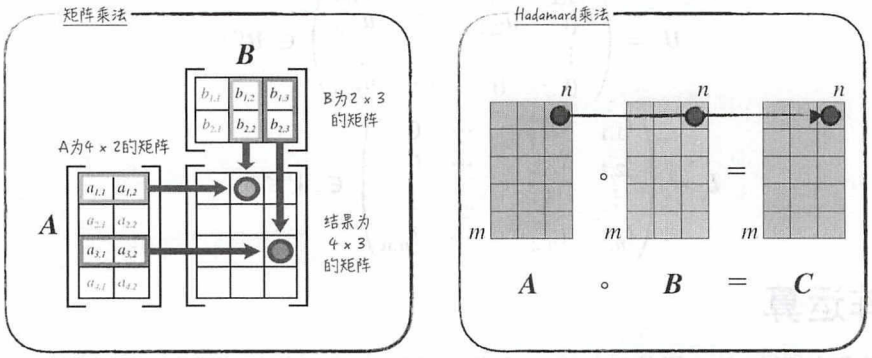
（1）矩阵与数字的乘法。与数字的乘法类似，任意一个实数都能和任意一个矩阵相乘。假设k为实数，则它与矩阵$\boldsymbol{X}$的乘法定义如下：

$$k\boldsymbol{X} = \begin{pmatrix} kx_{1,1} & \cdots & kx_{1,m} \\ \vdots & & \vdots \\ kx_{n,1} & \cdots & kx_{n,m} \end{pmatrix} = (kx_{i,j}) \in \mathbf{R}^{n \times m} \tag{3-7}$$

（2）矩阵与矩阵的乘法。它只有在第一个矩阵的列数和第二个矩阵的行数相同时才有定义。假设$\boldsymbol{A}$为$n \times p$的矩阵，$\boldsymbol{B}$为$p \times m$的矩阵，则它们之间的乘积为一个$n \times m$的矩阵，记为$\boldsymbol{AB}$[3]，具体的定义如下：

$$\boldsymbol{A} = (a_{i,j}) \in \mathbf{R}^{n \times p};\ \boldsymbol{B} = (b_{i,j}) \in \mathbf{R}^{p \times m}$$
$$\boldsymbol{AB} = \left(\sum_{r=1}^{p} a_{i,r} b_{r,j}\right) \in \mathbf{R}^{n \times m} \tag{3-8}$$

举个具体的例子：$\boldsymbol{A}$为4×2的矩阵，$\boldsymbol{B}$为2×3的矩阵，它们之间的乘法计算过程如图3-2a 所示。



（a）　　（b）

图 3-2[4]

不难证明，矩阵的乘法满足结合律，即$(\boldsymbol{AB})\boldsymbol{C} = \boldsymbol{A}(\boldsymbol{BC})$，以及分配律，即$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}$，但不满足交换律（在通常情况下，两个矩阵交换顺序后，乘法运算的要求都不再满足）。这一点与数字的乘法有很大的不同。另外，任何一个矩阵与单

[3] 在有的文献中，矩阵的乘法也被记为$\boldsymbol{A} \cdot \boldsymbol{B}$，但这种记法很容易和后面将介绍的向量内积发生混淆。因此本书并不采用这种记录方法。

[4] 图片参考自维基百科。

书籍是人类进步的阶梯

位矩阵的乘积（前提条件是矩阵乘法的要求被满足）等于其本身，比如$\boldsymbol{I}_n\boldsymbol{A} = \boldsymbol{A} = \boldsymbol{A}\boldsymbol{I}_p$。因此单位矩阵可以被看作矩阵中的 1。

在实际中，线性模型常常用矩阵乘法来表示。举个简单的例子，假设线性模型为：

$$\begin{cases} y_1 = ax_1 + b \\ y_2 = ax_2 + b \end{cases} \tag{3-9}$$

令$\boldsymbol{X} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix}$，$\boldsymbol{\beta} = \begin{pmatrix} a \\ b \end{pmatrix}$，$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$，则公式（3-9）可以表示为$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta}$。

（3）矩阵的 Hadamard 乘积（Hadamard product 或者 element-wise multiplication）。假设矩阵$\boldsymbol{A}, \boldsymbol{B}$同为 $n \times m$的矩阵，则它们之间的 Hadamard 乘积仍为$n \times m$的矩阵，记为$\boldsymbol{A} \circ \boldsymbol{B}$。计算过程如图 3-2b 所示，具体的公式如下：

$$\boldsymbol{A} \circ \boldsymbol{B} = (a_{i,j}b_{i,j}) \in \mathbf{R}^{n \times m} \tag{3-10}$$

矩阵的 Hadamard 乘积在数学上并不太常用，但在编程时，我们常用它来同时计算多组数据的乘积。

3．矩阵的除法：逆矩阵（inverse matrix）

（1）对矩阵求逆是专门针对方阵的，即行数等于列数的矩阵。假设$\boldsymbol{M}$是一个$n \times n$的矩阵，若存在一个$n \times n$的矩阵$\boldsymbol{N}$使得它们的乘积等于n阶单位矩阵，如公式（3-11）所示，则称矩阵$\boldsymbol{N}$为矩阵$\boldsymbol{M}$的逆矩阵，记为$\boldsymbol{M}^{-1}$，而$\boldsymbol{M}$则被称为可逆矩阵。

$$\boldsymbol{M}\boldsymbol{N} = \boldsymbol{N}\boldsymbol{M} = \boldsymbol{I}_n \tag{3-11}$$

（2）数学上可以证明，一个矩阵的逆矩阵如果存在，则逆矩阵唯一。所以对于方阵$\boldsymbol{M}$，如果存在另一个方阵$\boldsymbol{L}$，使得$\boldsymbol{M}\boldsymbol{L} = \boldsymbol{I}_n$，则一定有$\boldsymbol{L}\boldsymbol{M} = \boldsymbol{I}_n$，且$\boldsymbol{L} = \boldsymbol{N} = \boldsymbol{M}^{-1}$。

（3）关于逆矩阵，有如下几个常用的公式：

$$\begin{aligned} (\boldsymbol{M}^{-1})^{-1} &= \boldsymbol{M} \\ (k\boldsymbol{M})^{-1} &= \frac{1}{k}\boldsymbol{M}^{-1} \\ (\boldsymbol{M}\boldsymbol{N})^{-1} &= \boldsymbol{N}^{-1}\boldsymbol{M}^{-1} \end{aligned} \tag{3-12}$$

4．矩阵的转置（transpose）

（1）形象点理解，矩阵的转置就是将矩阵沿着对角线对调一下。假设$\boldsymbol{X}$为$n \times m$的矩阵，则它的转置为$m \times n$的矩阵，记为$\boldsymbol{X}^{\mathrm{T}}$。具体的公式如下：

$$\begin{aligned} \boldsymbol{X} &= (x_{i,j}) \in \mathbf{R}^{n \times m} \\ \boldsymbol{X}^{\mathrm{T}} &= (x_{j,i}) \in \mathbf{R}^{m \times n} \end{aligned} \tag{3-13}$$

（2）关于矩阵的转置，有如下几个常用的公式，其中假设k为实数，而且公式中涉及的矩阵乘法和逆矩阵都是有意义的。

$$\begin{aligned} (\boldsymbol{X}^{\mathrm{T}})^{\mathrm{T}} &= \boldsymbol{X} \\ (\boldsymbol{X} + \boldsymbol{Y})^{\mathrm{T}} &= \boldsymbol{X}^{\mathrm{T}} + \boldsymbol{Y}^{\mathrm{T}} \\ (k\boldsymbol{X})^{\mathrm{T}} &= k\boldsymbol{X}^{\mathrm{T}} \end{aligned}$$

书籍是人类进步的阶梯

$$
\begin{aligned}
(\boldsymbol{XY})^{\mathrm{T}} &= \boldsymbol{Y}^{\mathrm{T}}\boldsymbol{X}^{\mathrm{T}} \\
(\boldsymbol{X}^{\mathrm{T}})^{-1} &= (\boldsymbol{X}^{-1})^{\mathrm{T}}
\end{aligned}
\tag{3-14}
$$

3.1.4 代码实现

第三方库 NumPy 已经实现了上面所介绍的矩阵运算，程序清单 3-1 是具体的代码。读者可以打开 Python Shell 或者 IPython，跟随本书一起练习吧。

（1）Python 中主要有两种表示矩阵的方法。一种是 matrix 类，如第 5～8 行代码所示。另一种是二维 array，如第 9～12 行代码所示。这两种方法的主要差别在于默认乘法的不同，当矩阵是 matrix 类时，比如“A”，默认的乘法为矩阵的乘法，如第 13～18 行代码所示，由于“A”与自身不能相乘，程序报错；当矩阵是 array 类时，比如“B”，默认的乘法是 Hadamard 乘法，如第 19～22 行代码所示。在实际中，大多数情况下，我们使用后一种方法，也就是用二维 array 表示矩阵。

（2）第三方库 NumPy 还提供专门的函数用于创建特殊的矩阵，如第 23～35 行代码所示。

（3）对于一个创建好的矩阵，可以像二维数组那样提取其中的某个元素，也可以提取其中的某几行或某几列，如第 36～46 行代码所示。

程序清单 3-1 矩阵创建——NumPy

```
 1 | >>> import numpy as np
 2 | >>> from numpy.linalg import inv
 3 | >>>
 4 | >>> # 创建矩阵
 5 | >>> A = np.matrix([[1, 2], [3, 4], [5, 6]])
 6 | matrix([[1, 2],
 7 |         [3, 4],
 8 |         [5, 6]])
 9 | >>> B = np.array(range(1, 7)).reshape(3, 2)
10 | array([[1, 2],
11 |        [3, 4],
12 |        [5, 6]])
13 | >>> A * A
14 | Traceback (most recent call last):
15 |   File "<stdin>", line 1, in <module>
16 |   File "/Library/Frameworks/Python.framework/Versions/2.7/lib/python2.7/
                site-packages/numpy/matrixlib/defmatrix.py", line 343, in __mul__
17 |     return N.dot(self, asmatrix(other))
18 | ValueError: shapes (3,2) and (3,2) not aligned: 2 (dim 1) != 3 (dim 0)
19 | >>> B * B
20 | array([[ 1,  4],
21 |        [ 9, 16],
22 |        [25, 36]])
23 | >>> # 创建特殊矩阵
```

书籍是人类进步的阶梯


```
24 |  ... np.zeros((3, 2))
25 |  array([[ 0.,  0.],
26 |         [ 0.,  0.],
27 |         [ 0.,  0.]])
28 |  >>> np.identity(3)
29 |  array([[ 1.,  0.,  0.],
30 |         [ 0.,  1.,  0.],
31 |         [ 0.,  0.,  1.]])
32 |  >>> np.diag([1, 2, 3])
33 |  array([[1, 0, 0],
34 |         [0, 2, 0],
35 |         [0, 0, 3]])
36 |  >>> # 矩阵中向量的提取
37 |  >>> m = np.array(range(1, 10)).reshape(3, 3)
38 |  >>> # 提取行向量
39 |  >>> m[[0, 2]] # 或者m[[True, False, True]]
40 |  array([[1, 2, 3],
41 |         [7, 8, 9]])
42 |  >>> # 提取列向量
43 |  >>> m[:, [1, 2]] # 或者m[:, [False, True, True]]
44 |  array([[2, 3],
45 |         [5, 6],
46 |         [8, 9]])
```

（4）矩阵的加减法运算以及与数字的乘积运算和数字计算一样，直接用加减乘号表示，如程序清单 3-2 中的第 12～20 行代码所示。

（5）当矩阵用二维 array 表示时，直接的乘号表示 Hadamard 乘积，如第 22～24 行代码所示。矩阵的乘法需要使用 dot 函数，如第 26～28 行代码所示。

（6）矩阵的转置由 transpose 函数完成，如第 9～11 行代码所示，而逆矩阵由 inv 函数完成，如第 30～35 行代码所示。

程序清单 3-2 矩阵运算——NumPy

```
 1 |  >>> import numpy as np
 2 |  >>> from numpy.linalg import inv
 3 |  >>>
 4 |  >>> # 矩阵的计算
 5 |  >>> n = np.array(range(1, 5)).reshape(2, 2)
 6 |  >>> n
 7 |  array([[1, 2],
 8 |         [3, 4]])
 9 |  >>> np.transpose(n)
10 |  array([[1, 3],
11 |         [2, 4]])
12 |  >>> n + n
13 |  array([[2, 4],
14 |         [6, 8]])
15 |  >>> n - n
16 |  array([[0, 0],
17 |         [0, 0]])
```

书籍是人类进步的阶梯


```
>>> 3 * n
array([[ 3,  6],
       [ 9, 12]])
>>> ## Hadamard乘积
>>> n * n
array([[ 1,  4],
       [ 9, 16]])
>>> ## 矩阵乘法
>>> n.dot(n)
array([[ 7, 10],
       [15, 22]])
>>> ## 矩阵的逆矩阵
>>> inv(n)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
>>> np.dot(inv(n), n)
array([[  1.00000000e+00,   0.00000000e+00],
       [  1.11022302e-16,   1.00000000e+00]])
```

3.1.5 向量空间

向量空间（vector space）是一个比较深刻的数学概念。为了便于理解，我们抛开复杂的公理化定义[5]，从直观上来理解向量空间的相关知识。

我们生活的世界在空间上是一个三维空间[6]，在这个现实世界里建立长宽高坐标系，也就是x, y, z坐标系。这个现实中的每一个点都能被表示成一个三维行向量，如图 3-3a 所示。从数学上来看，任意一个三维的行向量$\boldsymbol{X} = (x_1, x_2, x_3)$，可以被写为$\boldsymbol{X} = x_1(1,0,0) + x_2(0,1,0) + x_3(0,0,1)$。用学术一些的话来表述就是任意一个三维行向量可以被$\boldsymbol{e}_1 = (1,0,0), \boldsymbol{e}_2 = (0,1,0), \boldsymbol{e}_3 = (0,0,1)$这 3 个行向量线性表示，而且反过来，$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$这

---

[5] 向量空间的公理化的定义如下（来源于维基百科）。给定域F上的向量空间是一个集合V，其上定义了两种二元运算。

向量加法：$+: V \times V \to V$，把V中的两个元素 $\boldsymbol{u}$ 和 $\boldsymbol{v}$ 映射到V中另一个元素，记作$\boldsymbol{u} + \boldsymbol{v}$。

标量乘法：$\cdot: F \times V \to V$，把F中的一个元素 a 和 V 中的一个元素$\boldsymbol{u}$变为V中的另一个元素，记作$a \cdot \boldsymbol{u}$。

V中的元素称为向量，相对地，F中的元素称为标量。而集合V上的两个运算满足下面的公理。

向量加法结合律：$\boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) = (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}$。

向量加法交换律：$\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{v} + \boldsymbol{u}$。

存在向量加法的单位元：V里存在一个叫作零向量的元素，记作$\boldsymbol{0}$，使得对任意$\boldsymbol{u} \in V$，都有$\boldsymbol{u} + \boldsymbol{0} = \boldsymbol{u}$。

向量加法的逆元素：对任意$\boldsymbol{u} \in V$，都存在$\boldsymbol{v} \in V$，使得$\boldsymbol{u} + \boldsymbol{v} = \boldsymbol{0}$。

标量乘法对向量加法满足分配律：$a \cdot (\boldsymbol{v} + \boldsymbol{w}) = a \cdot \boldsymbol{v} + a \cdot \boldsymbol{w}$。

标量乘法对域加法满足分配律：$(a + b) \cdot \boldsymbol{v} = a \cdot \boldsymbol{v} + b \cdot \boldsymbol{v}$。

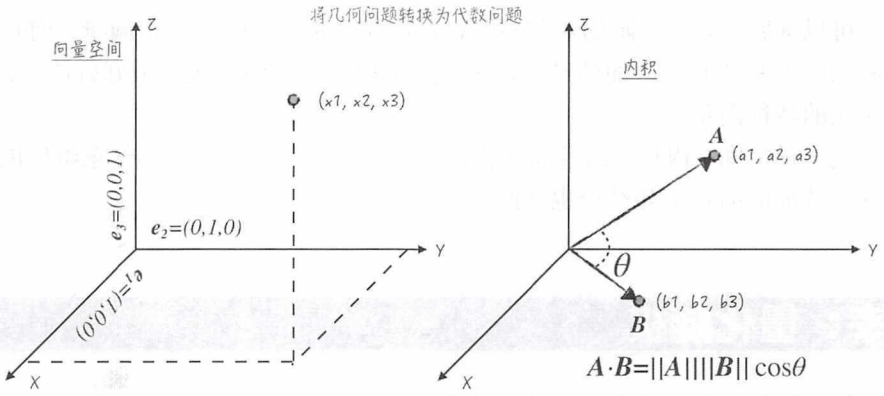
标量乘法与标量的域乘法相容：$a(b \cdot \boldsymbol{v}) = (ab) \cdot \boldsymbol{v}$。

标量乘法有单位元：域F的乘法单位元“1”满足对任意$\boldsymbol{v}$，$1 \cdot \boldsymbol{v} = \boldsymbol{v}$。

[6] 按照爱因斯坦的相对论，我们生活的世界是四维向量空间的。除了空间三维外，还有一个维度是时间。

书籍是人类进步的阶梯

3 个行向量的任意线性组合都对应着现实空间中的某一点。像这样的空间被称为向量空间，而$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$被称为**向量空间的基**。不难看出，其实$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$分别对应着$x$轴、$y$轴和$z$轴。



（a）　　（b）

图 3-3

针对三维向量空间，定义行**向量的内积**（dot product）。假设行向量$\boldsymbol{A} = (a_1, a_2, a_2)$和行向量$\boldsymbol{B} = (b_1, b_2, b_3)$，它们之间的内积定义为：

$$\boldsymbol{A} \cdot \boldsymbol{B} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{3-15}$$

其实$\boldsymbol{A}, \boldsymbol{B}$之间的内积可以表达为矩阵乘法，即$\boldsymbol{A} \cdot \boldsymbol{B} = \boldsymbol{A}\boldsymbol{B}^{\mathrm{T}}$。数学上可以证明公式(3-16)，其中$\|\boldsymbol{A}\| = \sqrt[2]{(a_1^2 + a_2^2 + a_3^2)}$为点$A$到原点的距离，$\|\boldsymbol{B}\|$的定义类似；$\theta$为向量$\boldsymbol{A}, \boldsymbol{B}$之间的夹脚，如图 3-3b 所示。注意到当$\boldsymbol{B}$到原点的距离等于 1 时，$\boldsymbol{A} \cdot \boldsymbol{B}$就是向量$\boldsymbol{A}$在向量$\boldsymbol{B}$方向上投影的长度：

$$\boldsymbol{A} \cdot \boldsymbol{B} = \|\boldsymbol{A}\|\|\boldsymbol{B}\| \cos\theta \tag{3-16}$$

有了内积定义后，就可以更加数理化地定义向量间的**正交关系**（对应到空间上，就是两个向量相互垂直）。若两个行向量$\boldsymbol{X}, \boldsymbol{Y}$的内积等于 0，则称它们是正交的。容易得到，上面提到的基$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$是相互正交的，被称为正交基。

既然行向量可以被看作三维空间里的一个点，那么$n \times 3$的矩阵$\boldsymbol{X}$可以被看作三维空间里点的集合。现在我们想在这个空间里找到一条直线，使得矩阵中的点在这条直线的投影之和最长。结合内积的定义，假设$\boldsymbol{w}$为行向量，将上面的目标表示为数学公式，其中向量$\boldsymbol{v}$是长度为 1 的向量，表示直线的方向。

$$\boldsymbol{v} = \operatorname{argmax}_{\|\boldsymbol{w}\|=1} \left\|\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}\right\|^2 = \operatorname{argmax}_{\|\boldsymbol{w}\|=1} \boldsymbol{w}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}} \tag{3-17}$$

为了解决这个问题，需要引入另一个很复杂的数学概念：**特征向量**（eigenvector）和**特征值**（eigenvalue）。假设$\boldsymbol{M}$是一个三阶对称矩阵，即$\boldsymbol{M}^{\mathrm{T}} = \boldsymbol{M}$（注意到公式(3-17)里，$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$就是

书籍是人类进步的阶梯

一个三阶对称矩阵）。满足下面条件的非零行向量$\boldsymbol{w}$被称为$\boldsymbol{M}$的特征向量，对应的λ被称为特征值[7]。

$$\boldsymbol{M}\boldsymbol{w}^{\mathrm{T}} = \lambda \boldsymbol{w}^{\mathrm{T}} \tag{3-18}$$

数学上可以证明，对于三阶的对称矩阵，存在 3 个相互正交的特征向量，而它们可以组成三维空间的一组基。有了上面的结论，公式（3-17）定义的问题就很好解决了：$\boldsymbol{v}$就是最大特征值对应的特征向量[8]。

上面有关向量空间、内积、向量间夹脚以及正交的定义可以推广到任意维度的空间。有关特征向量、特征值的定义和结论也如此。

3.2 概率：量化随机

概率是量化事物随机性或者可能性的数学工具，在很多领域都有广泛的应用。但遗憾的是，概率或者随机本身是数学里人类理解最差的分支。在通常情况下，可以将概率从直观上理解为事件发生的比例。如图 3-4 所示，向图中的方形框随机投掷小球，那么小球落入圆圈的概率就等于圆圈的面积除以方形框的面积。

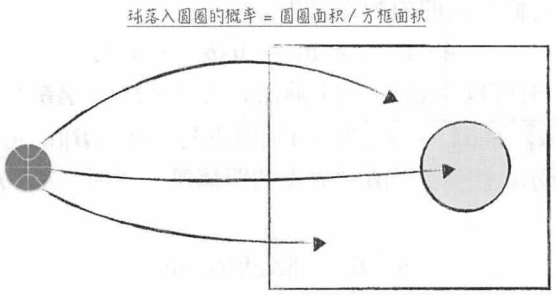


图 3-4

[7] 在数学上，特征向量和特征值并非只定义在对称矩阵上。事实上对于任意一个方阵$\boldsymbol{M}$，满足下面公式的向量和实数分别为$\boldsymbol{M}$的特征向量和特征值：

$$\boldsymbol{M}\boldsymbol{w}^{\mathrm{T}} = \lambda \boldsymbol{w}^{\mathrm{T}}$$

当然，对于一般的方阵，特征向量相互正交的结论就不再成立了。

[8] 这里给出简易的证明：假设$\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3$是对称矩阵$\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$的特征向量，且长度都为 1。它们相互正交，对应的特征值为$\lambda_1, \lambda_2, \lambda_3$。且有$\lambda_1 \geqslant \lambda_2 \geqslant \lambda_3$。任何一个长度为 1 的行向量$\boldsymbol{w}$可以写为$\boldsymbol{w} = a\boldsymbol{w}_1 + b\boldsymbol{w}_2 + c\boldsymbol{w}_3$，其中$a^2 + b^2 + c^2 = 1$，则

$$\boldsymbol{w}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}} = \lambda_1 a^2 + \lambda_2 b^2 + \lambda_3 c^2$$

$$\boldsymbol{w}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}} \leqslant \lambda_1(a^2 + b^2 + c^2) = \lambda_1$$

因此当$\boldsymbol{w}$等于$\boldsymbol{w}_1$时，点到直线的投影之和最大，即$\boldsymbol{w}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w}^{\mathrm{T}}$达到最大值，所以$\boldsymbol{v} = \boldsymbol{w}_1$。

书籍是人类进步的阶梯

概率对数据科学尤其重要，举两个常见的例子：在搭建模型时，带有一些随机性的模型和算法往往预测效果会好于完全确定性的模型和算法；在异常检测时，理解概率能帮助我们区分真正的异常和正常情况下的随机扰动。

本节将着重介绍数据科学中常用到的概率知识，帮助读者在之后的章节里更好地掌握与概率相关的模型。

3.2.1 定义概率：事件和概率空间

我们首先从掷骰子这个常见的例子中引出概率的定义。假设我们连续随机地掷两次骰子，并计算两次所得点数的和。记第一次掷骰子得到的点数为X_1，第二次的点数为X_2，两次点数之和为$XX = X_1 + X_2$。容易得到XX可能的取值为 2～12。将$XX = i$记为事件E_i。但其实上面列举的事件还可以划分为更加细小的随机样本，比如$XX = 3$对应的事件E_3可以分解为两个事件，一是第一次点数是 1，第二次点数是 2，记为（1，2）；二是第一次点数是 2，第二次点数是 1，记为（2，1）。整个过程如图 3-5 所示。将事件E发生的概率记为$P(E_i)$，则

$$P(E_3) = P((1,2) \cup (2,1)) = P((1,2)) + P((2,1)) = 1/6 \tag{3-19}$$

对于其他结果也类似地定义它们发生的概率。

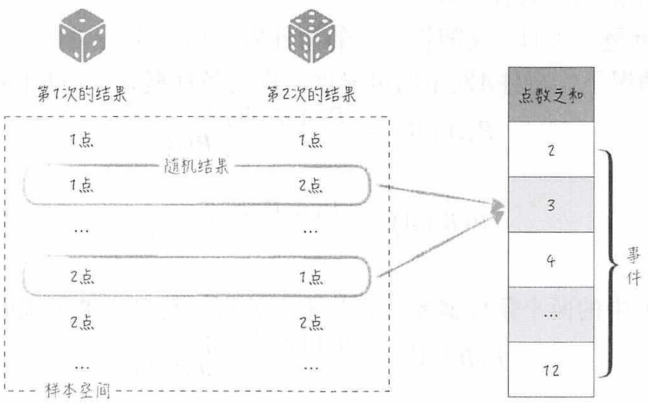


图 3-5

将上面例子中的方法推广到一般情况，给出概率的定义[9]。

[9] 正文中给出的定义并不是严格意义上的公理化的定义。概率严格定义为，它是定义在概率空间上一种度量，也就是从样本事件到实数的函数。这个函数满足所谓的柯尔莫果洛夫公理（Kolmogorov Axioms）。具体地，假设P为概率：

- 对于任意一个事件E，则$P(E) \geqslant 0$；
- 对于所有可能事件的集合Ω，则$P(\Omega) = 1$；
- 任意两两互不相交的事件可数序列$E_1, E_2, \ldots$，则$P(E_1 \cup E_2 \cup \ldots) = \sum P(E_i)$。

书籍是人类进步的阶梯

将所有不能再分的随机结果，记为ω，放在一起组成一个可数的非空集合，这个集合就叫作**样本空间**（sample space），记为S。样本空间里的子集被称为事件。而概率是一个定义在样本空间上的实数函数，记为P，它满足下面两个条件：

- $P(\omega) \geqslant 0$，对于所有的ω都成立；
- $\sum_{\omega \in S} P(\omega) = 1$。

对于一个事件 E，对应的概率为$P(E) = \sum_{\omega \in E} P(\omega)$。一个样本空间加上在其基础上定义的概率就成为一个**概率空间**。根据概率的定义，可以得到如下的公式，其中A, B均为随机事件，而A^c表示事件A的补集。

$$0 \leqslant P(A) \leqslant 1$$

$$P(A^c) = 1 - P(A)$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{3-20}$$

3.2.2 条件概率：信息的价值

上面讨论了单个事件和多个事件发生一个的概率。现在将讨论两个或多个事件同时发生的概率。假设事件A和B是两个不同的事件，它们俩同时发生的概率为$P(A \cap B)$。这个概率与两个事件单独发生的概率有什么联系吗？

为了解答这个问题，我们定义随机事件的条件概率如公式（3-21）所示。其中$P(A \mid B)$表示在事件B发生的情况下，事件A发生的可能性，称为**条件概率**。$P(B \mid A)$的含义类似。

$$P(A \mid B) = {P(A \cap B)}/{P(B)}$$

$$P(B \mid A) = {P(A \cap B)}/{P(A)} \tag{3-21}$$

将公式（3-21）中的两个条件概率结合起来，就可以得到所谓的贝叶斯定理：

$$P(B \mid A) = {P(A \mid B)P(B)}/{P(A)} \tag{3-22}$$

举一个简单的例子来直观地感受条件概率这个概念。假设在一个大学的班级里：

- 来自重庆的学习比例为 10%，而这批学生中喜欢吃辣的比例为 90%；
- 剩下的来自其他地区的学生占比 90%，他们当中喜欢吃辣的比例为 30%。

为了表述清楚，用A表示某学生来自重庆，B表示该学生喜欢吃辣。根据上面的描述，在没有任何其他信息的条件下，一个学生来自重庆的概率为 10%，即$P(A) = 0.1$。但如果我们知道了这个学生喜欢吃辣，那么显然他来自重庆的比例会上升，因为重庆人更喜欢吃辣。也就是说，喜欢吃辣这条信息对判断他是否来自重庆是有价值的，但应该如何量化它呢？我

书籍是人类进步的阶梯

书籍是人类进步的阶梯

们可以通过条件概率来量化吃辣这条信息的价值。具体地，根据贝叶斯定理可以得到 $P(A \mid B) = 0.25$[10]，如图 3-6 所示。通俗来讲就是，知道这个学生喜欢吃辣后，他来自重庆的概率从 10%上升到了 25%。这就是从信息中得到的价值。

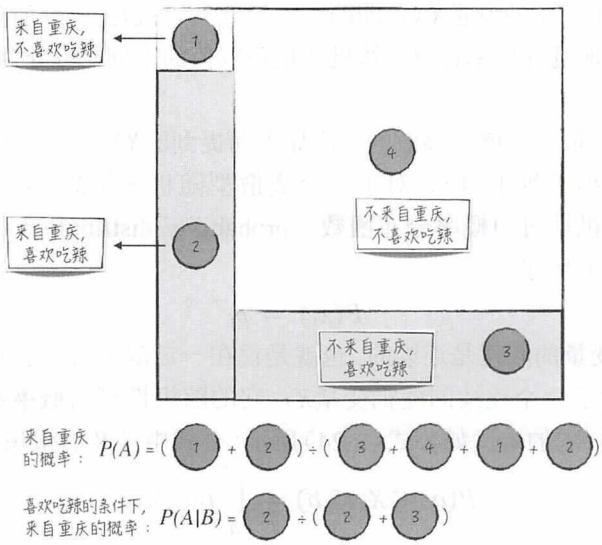


图 3-6

上面的例子告诉我们，条件概率$P(A \mid B)$和事件原本概率$P(A)$之间的差异体现了发生事件B这条信息对事件A是否发生的价值。这部分的细节将在第 9 章里展开，这里就先不做过多的讨论。

如果条件概率等于原本的概率，即$P(A \mid B) = P(A)$（在这种条件下，很容易推出$P(B \mid A) = P(B)$），则称事件A, B**相互独立**。换句话说，事件B与事件A毫无联系，前者发生与否不会影响后者的发生。

当两个事件相互独立时，可以推出$P(A \cap B) = P(A)P(B)$。在此基础上，定义任意多个相互独立的事件：假设$A_1, A_2, \ldots, A_n$是一系列随机事件，这些事件都是相互独立的当且仅当对其任一有限子集$A_{i1}, A_{i2}, \ldots, A_{in}$，都满足$P(A_{i1} \cap \ldots \cap A_{in}) = P(A_{i1}) \ldots P(A_{in})$。

---

[10] 这里给出具体的计算过程。首先将事件B的概率分解，即$P(B) = P(B \cap A) + P(B \cap A^c)$。再根据条件概率的公式可以得到

$$P(B) = P(B \mid A)P(A) + P(B \mid A^c)P(A^c)$$

$$P(A \mid B) = {P(B \mid A)P(A)}/{[P(B \mid A)P(A) + P(B \mid A^c)P(A^c)]}$$

根据问题的描述，我们有$P(A) = 0.1, P(A^c) = 0.9, P(B \mid A) = 0.9, P(B \mid A^c) = 0.3$。代入公式计算可以得到 $P(A \mid B) = {0.09}/{(0.09 + 0.27)} = 0.25$。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

3.2.3 随机变量：两种不同的随机

将随机事件进一步量化，在其基础上定义随机变量：将随机事件映射为数字（常为实数）的函数[11]。比如 3.2.1 节中的变量XX，即两次点数之和，就是一个随机变量。在随机变量的基础上，可以更方便地进行概率计算。随机变量按取值的不同，可分为**离散型随机变量**和**连续型随机变量**。

- 离散型随机变量的取值是离散的，比如上面提到的XX，它可能的取值为离散的自然数，大于等于 2 且小于等于 12。对于一个离散型随机变量X，假设它可能的取值记为$x_1, x_2, \ldots, x_n$。X的随机性可由**概率分布函数**（probability distribution function）描述，具体的定义如公式（3-23）所示。

$$P(x_i) = p_i \tag{3-23}$$

- 连续型随机变量的取值是连续的。也就是说在一定范围内，它可以是其中的任意值，比如人体的身高。对于一个连续的随机变量X，它的随机性可由**概率密度函数**（probability density function）描述，它的定义如公式（3-24）所示（公式中涉及的微积分请参考第 3.3 节）。

$$P(a \leqslant X \leqslant b) = \int_a^b f_X(x)\mathrm{d}x$$
$$f_X(x) = \frac{\mathrm{d}}{\mathrm{d}x}P(-\infty \leqslant X \leqslant x) \tag{3-24}$$

在随机变量X的基础上，有如下几个常用的函数和统计指标。

- **累积分布函数**（cumulative distribution function，CDF）的定义如下：

$$F_X(x) = P(X \leqslant x) \tag{3-25}$$

- **期望**（expected value）。这个统计量可以被直观地理解为随机变量的加权平均值，通常记为$E[X]$。具体的计算公式如下（如果期望存在）：

$$E[X] = \begin{cases} \sum p_i x_i, & X\text{是离散型随机变量} \\ \int x f_X(x)\mathrm{d}x, & X\text{是连续型随机变量} \end{cases} \tag{3-26}$$

- **方差**（variance），记为$\mathrm{Var}(X)$，用于度量随机变量的分散情况。它的定义公式为（如果方差存在）：

$$\mathrm{Var}(X) = E[(X - E[X])^2] = E[X^2] - (E[X])^2 \tag{3-27}$$

- **协方差**（covariance），记为$\mathrm{Cov}(X, Y)$，用于度量两个随机变量的整体变化幅度和它们之间的相关关系。它的定义公式如公式（3-28）所示。容易看到，随机变量的方差是一种特殊的协方差。

---

[11] 严格的数学定义要求函数为可测函数。可测函数是测度论里面的一个数学概念，比较复杂且与数据科学关系不大，因此不做展开。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

$$\mathrm{Cov}(X,Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y] \tag{3-28}$$

对于随机变量的期望和方差，有如下几个常用的公式：

$$E[aX + bY] = aE[X] + bE[Y]$$

$$\mathrm{Cov}(X,X) = \mathrm{Var}(X)$$

$$\mathrm{Var}(aX + bY) = a^2\mathrm{Var}(X) + b^2\mathrm{Var}(Y) + 2ab\mathrm{Cov}(X,Y) \tag{3-29}$$

讨论完单个随机变量的概率分布，下面将讨论两个随机变量间的条件概率分布。假设讨论的两个随机变量分别为X和Y。

- 若X和Y都是离散型随机变量，则它们之间的条件分布用条件概率分布函数来描述，具体的定义如下：

$$P(X = x_i \mid Y = y_j) = P(X = x_i, Y = y_j) \big/ P(Y = y_j) \tag{3-30}$$

- 若X和Y都是连续型随机变量，则它们之间的条件分布用条件概率密度函数来描述，具体的定义如下。其中，$f_{X,Y}$是变量X和Y的联合概率密度函数[12]，f_X是变量X的概率密度函数。

$$f_{Y|X}(y \mid X = x) = f_{X,Y}(x,y) \big/ f_X(x) \tag{3-31}$$

- 若X和Y中有一个是离散型随机变量，另一个是连续的，则按上面的方式类似地定义。严格的数学定义涉及的实变函数比较复杂，而且在实际中也不常接触到，因此就不做详细讨论了。

同相互独立的随机事件类似，下面将定义相互独立的随机变量，它在数据科学的模型中使用非常广泛。

对于一个随机变量X和一个实数a，X是否小于a就定义了一个随机事件，记为$[X < a]$。然后基于这些随机事件的相互独立性，就可以定义随机变量是否独立。具体地，假设$X_1, X_2, \ldots, X_n$是一系列随机变量，对于其中任一有限子集$X_{i1}, X_{i2}, \ldots, X_{in}$以及任意数字子集$a_{i1}, a_{i2}, \ldots, a_{in}$，若随机事件$[X_{i1} \leqslant a_{i1}], [X_{i2} \leqslant a_{i2}], \ldots, [X_{in} \leqslant a_{in}]$是相互独立的，则随机变量$X_1, X_2, \ldots, X_n$是相互独立的。

如果随机变量是相互独立的，则它们的概率分布和统计指标一些比较简便的计算公式，如公式（3-32）所示。为了表述简洁，这里仅以两个独立的随机变量为例（记为X, Y），多个随机变量的情况类似。

$$P(X = x_i, Y = y_j) = P(X = x_i)P(Y = y_j) \text{或} f_{X,Y} = f_X f_Y$$

---

[12] 联合概率密度函数的定义如下：假设X和Y是两个连续型随机变量。若函数$f_{X,Y}$满足下面的公式，则称$f_{X,Y}$为随机变量X和Y的联合概率密度函数。

$$P(a \leqslant X \leqslant b, c \leqslant Y \leqslant d) = \int_c^d \int_a^b f_{X,Y}\,\mathrm{d}x\mathrm{d}y$$

书籍是人类进步的阶梯

书籍是人类进步的阶梯

$$E[XY] = E[X]E[Y]$$
$$\mathrm{Var}(aX + bY) = a^2\mathrm{Var}(X) + b^2\mathrm{Var}(Y) \tag{3-32}$$

3.2.4 正态分布：殊途同归

本节将讨论一个非常重要的概率分布：**正态分布**（normal distribution，也称为高斯分布 Gaussian distribution）。若随机变量X服从正态分布，则它是一个连续型随机变量，相应的概率密度函数如下：

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \tag{3-33}$$

公式（3-33）中的μ, σ^2是概率分布的参数，可以证明随机变量X的期望等于μ，方差等于σ^2，如公式（3-34）所示。通常将其记为$X \sim \mathcal{N}(\mu, \sigma^2)$。

$$E[X] = \mu, \mathrm{Var}(X) = \sigma^2 \tag{3-34}$$

当$\mu = 0, \sigma^2 = 1$时，我们称其为标准正态分布。可以证明随机变量$(X - \mu)/\sigma$服从标准正态分布，即$(X - \mu)/\sigma \sim \mathcal{N}(0, 1)$。不同参数下，正态分布的概率密度函数曲线如图 3-7 所示，参数μ将决定曲线的中心位置，而参数σ^2决定曲线的平坦程度。这个值越大，概率密度曲线越平坦。

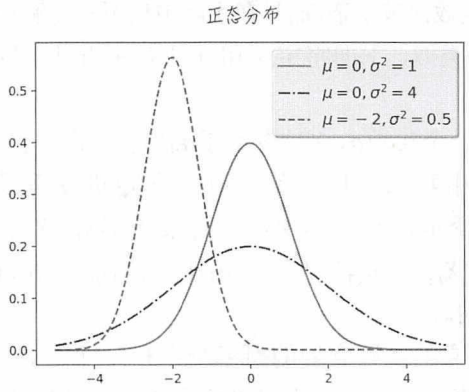


图 3-7

在实际中，我们发现有很多随机变量都大致服从正态分布，因此这个分布的应用非常广，比如在搭建模型时，通常会假设模型的随机扰动项服从正态分布[13]。为什么正态分布会如此广泛地存在呢？下面给出一个比较合理的猜测：**中心极限定理**（central limit theorem）。

[13] 这个假设并不总是符合现实情况，这会导致模型的效果并不好，甚至会模型得出错误的结论。很多模型的改进和创新也是基于对这一假设改进，相关的讨论将后面章节展开。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

假设随机变量$X_1, X_2, \ldots, X_n$独立同分布[14]（independent and identically distributed ，缩写为 i.i.d），且具有有限的期望和方差，记为$E[X_i] = m, \mathrm{Var}(X_i) = v^2$。数学上可以证明如下的定理：

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_n$$
$$T_n = \sqrt{n}\left(\bar{X} - m/_v\right)$$
$$\lim_{n\to\infty} T_n \sim \mathcal{N}(0,1) \tag{3-35}$$

公式（3-35）表示在一定条件下，不管随机变量的分布如何，它们的和经过一定的线性变换后会逼近一个标准正态分布。可以形象地理解为，一定量的随机效应叠加起来就近似服从正态分布。下面以图 3-8 为例：一个骰子的点数近似于均匀分布；两个骰子点数之和的分布曲线近似于一个等边三角形；随着骰子数量的增多，点数之和的分布曲线就越来越接近于一个正态分布。

在现实中，很多观察到的随机变量实际上正是多个独立同分布的随机变量叠加起来的结果，因此根据中心极限定理，它会大致服从一个正态分布。

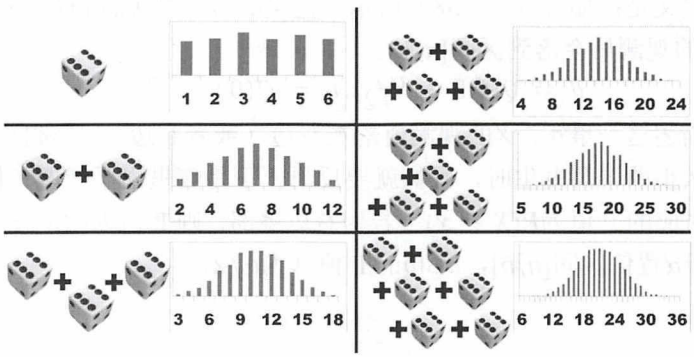


图 3-8[15]

3.2.5 P-value：自信的猜测

我们以正态分布为例，讨论一个在数学上很简单，但在统计学里应用非常广泛的一个概念：P 值（P-value）。P 值在数学上对应着**分位数方程**（quantile function），先来看看这个数学概念。

不妨设X是一个实数随机变量，而p是(0, 1)区间内的一个实数。则它的分位数方程定义为：

[14] 如果随机变量$X_1, X_2, \ldots, X_n$不相互独立，则中心极限定理不再成立。可以考虑如下的反例：假设X_1是均匀的-1，1 分布，即$P(X_1 = -1) = P(X_2 = 1) = 0.5$。而对于$i > 1$，随机变量$X_i = X_1$，那么$X_i$也是均匀的$-1$，1 分布。但$\frac{1}{n}\sum_{i=1}^{n} X_i$等于 1 或者$-1$。显然这个概率分布不会逼近正态分布。

[15] 图片来源于 Jody Muelaner 博士的个人主页。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

$$Q(p) = \inf\{x \in \mathbf{R}, p \leqslant P(X \leqslant x)\} \tag{3-36}$$

即$Q(p)$为累积概率大于等于p值的最小实数。举一个简单的例子来展示这个定义，假设X是一个骰子的点数，那么它在1～6的自然数上均匀分布，即$P(X = i) = 1/6, i = 1,\ldots,6$。如果$p = 0.5$，比较容易可以得到：$P(X \leqslant 3) = 0.5$、$P(X \leqslant 4) = 4/6$以及$P(X \leqslant 2) = 2/6$，所以$Q(0.5) = 3$。以此类推，可以得到如下的分位数方程：

$$Q(p) = \begin{cases} 1, & 0 < p \leqslant 1/6 \\ 2, & 1/6 < p \leqslant 2/6 \\ 3, & 2/6 < p \leqslant 3/6 \\ 4, & 3/6 < p \leqslant 4/6 \\ 5, & 4/6 < p \leqslant 5/6 \\ 6, & 5/6 < p < 1 \end{cases} \tag{3-37}$$

介绍完数学公式，我们再来看看定义P值背后的思路。对于一个服从正态分布的随机变量X，它的观测值大多会落在期望周围，因为正态分布的概率密度在期望附近更大，观测值落在这个区域是更正常的事情，如图3-9所示。由此定义X的α**置信区间**（α通常等于0.95或者0.99）：概率等于α且以期望为中心的对称区域。

严谨的数学定义是：如公式（3-38）所示，定义a和b，则X的α置信区间为$[a, b]$。也就是有α的概率，X的观测值会落到区间$[a, b]$。

$$a = Q(0.5 + \alpha/2), b = Q(0.5 - \alpha/2) \tag{3-38}$$

换个角度来描述这个事实：X的观测值落在左边（或者右边）“尾部区域”是非常少见的情况。当概率这么小的事情发生时，我们就要反思一下是哪里出了问题。假设观测值为x，定义这个观测值对应的P值为$P(X \geqslant x)$（若为右边尾部，则P值为$P(X \leqslant x)$），如图3-9所示。容易得到对于α置信区间$[a, b]$，a和b的P值都为$\alpha/2$。

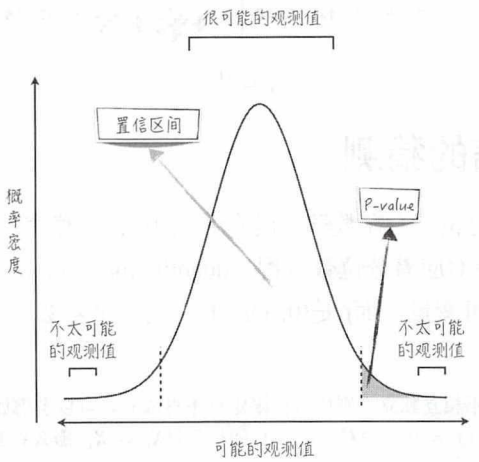


图3-9

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

在统计分析中，常用 P 值以及相应的置信区间来进行假设检验。相关内容将在第 4、5、7 章中详细讨论。

3.3 微积分

微积分是高等数学的基础，也是现代数学的开端。正如其名字，微积分主要包含两个互补的方面：**导数**（微分）和**积分**。导数主要研究函数在局部的变化速率，比如根据物体的位置函数求其移动速度。积分与之相反，常被用于计算函数在一段范围内的累积效应。比如在 3.2 节中，针对连续型随机变量，根据它的概率密度函数，计算随机变量落在某区间内的概率。在数据科学领域，积分更多地被用于理论研究，接触的机会较少；而导数则被大量使用，主要用于工程实现（求解最优化问题）。因此后者是本节介绍的重点。

3.3.1 导数和积分：位置、速度

与前文内容类似，本节依然通过一个例子来引入微积分的基本概念。这个例子读者都无比熟悉：行走（是的，在走路时，我们就在接触微积分）。为了易于理解，假设我们研究的对象叫小明，小明的行走路线是直线。行走的过程中会涉及两个量：位置和速度，记$l(t)$为t时刻离起点的距离，而$v(t)$为t时刻的速度。这两个函数是相互关联的，速度是位置的瞬间变化，而位置是速度在一段时间内的积累。微积分解决的就是如何从函数$l(t)$中导出函数$v(t)$（导数），以及如何从函数$v(t)$中导出函数$l(t)$（积分），如图 3-10 所示。

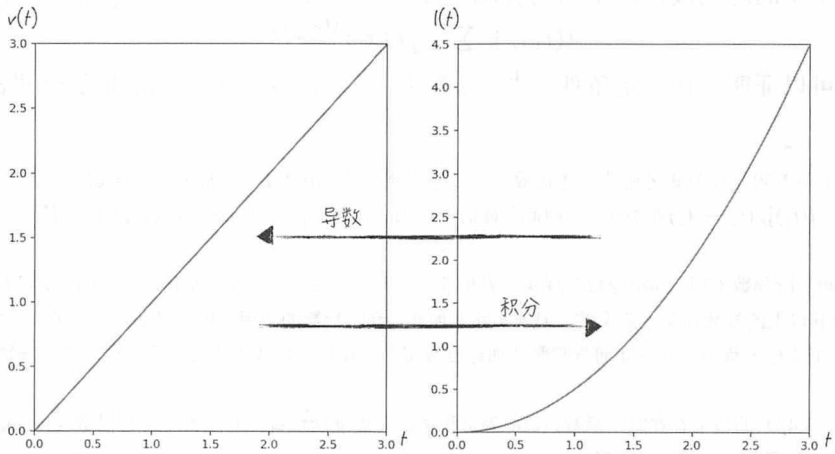


图 3-10

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

当已知时刻t_1的位置$\mathrm{d}(t_1)$以及时刻t_2的位置$l(t_2)$时，可以很容易地得出这段时间内的平均速度为$\bar{v} = [l(t_2) - l(t_1)]/(t_2 - t_1)$。当小明匀速前进时，那么这段时间内任一时刻的速度$v(t), t_1 \leqslant t \leqslant t_2$都等于这个平均速度$\bar{v}$。如果小明不是匀速前行呢？那么$t_1$时刻的速度就和平均速度存在一定的差异，但显然时间间隔越短，平均速度$\bar{v}$离t_1的速度$v(t_1)$也就越接近。当时间间隔达到极限 0 时，$\bar{v}$就会等于 $v(t_1)$。用数学语言表述就是平均速度的极限等于起点时刻的速度，如公式（3-39）所示[16]。

$$\lim_{t_2 \to t_1} [l(t_2) - l(t_1)] \big/ (t_2 - t_1) = \lim_{t_2 \to t_1} \bar{v} = v(t_1) \tag{3-39}$$

这个计算过程在数学上被称为求函数的导数，记为$l'(t) = v(t)$。导数的物理含义是非常明确的，当自变量t变化很小的量$\mathrm{d}t$时（在数学上，$\mathrm{d}t$被称为无穷小量），因变量$\mathrm{d}(t)$的变化幅度大约为$v(t)\mathrm{d}t$，记为$\mathrm{d}l(t) = l'(t)\mathrm{d}t$。在数学上，上面的公式叫作微分。

我们可以对速度函数$v(t)$继续求导，得到$v(t)$的一阶导数，同时也是$\mathrm{d}(t)$的二阶导数，它表示小明前进的加速度$a(t) = v'(t) = l''(t)$。以此类推可以定义函数$l(t)$的n阶导数[17]，记为$l^n(t)$。

类似地，当已知小明时刻t_1的速度等于$v(t_1)$，位置等于$\mathrm{d}(t_2)$时，假设小明是匀速前进的，那么t_2时刻，小明所在的位置是$l(t_2) = l(t_1) + v(t_1)(t_2 - t_1)$。如果小明不是匀速前进，显然按匀速前进估算的位置（即$l(t_1) + v(t_1)(t_2 - t_1)$）和实际位置（即$l(t_2)$）存在一定误差。那应该如何减少这个误差呢？我们考虑将时间区间$[t_1, t_2]$分为$n + 1$段，不妨设相应的分段时间点为$x_0, x_1, \ldots, x_n$，具体的计算公式如下：

$$x_0 = t_1;\ x_{i+1} = x_i + \frac{(t_2 - t_1)}{n} \tag{3-40}$$

在每一个小的时间段内，小明的移动可以近似为匀速前进。那么t_2时刻估算的位置为：

$$l(t_1) + \sum_{i=0}^{n} v(x_i) \frac{(t_2 - t_1)}{n} \tag{3-41}$$

数学上可以证明，在一定条件下[18]，n越大，公式（3-39）得到的估算结果离$l(t_2)$就越

---

[16] 当然公式（3-39）和公式之前的表述在数学上并不严谨，因为函数$l(t)$可能并不是可微函数。也就是说，极限$\lim_{t_2 \to t_1} [l(t_2) - l(t_1)]/(t_2 - t_1)$并不存在。因此严谨的表述为，如果$l(t_1)$是可微函数，则起点时刻的速度等于平均速度的极限。

[17] 与一阶和二阶导数不同，高阶导数在现实世界里的意义其实很模糊。正因为如此，微积分的发明者之一牛顿先生曾宣称，三阶及其以上的导数都是无意义的。但现实并非如此，高阶导数的应用范围还是非常广泛的。比如 1972 年美国总统竞选期间，尼克松号称他在任内让通货膨胀的加速度减缓。这让尼克松成为历史上第一位用三阶导数证明自己才能的国家领导人。

[18] 当公式（3-42）的极限存在时，函数$v(t)$称为可积函数。更准确地说，它是黎曼可积函数（Riemann integral）。在数据科学领域，常遇到的函数都是可积可微的函数。

在数学上，除了黎曼积分外，常用的积分还有勒贝格积分（Lebesgue integral）。这个积分的定义涉及实变函数理论，比较复杂，而且它在数据科学领域多用于理论研究，因此在这里就不做讨论了。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

近。当n趋近于正无穷大时，这两者就完全相等了。

$$l(t_2) - l(t_1) = \lim_{n\to\infty}\sum_{i=0}^{n} v(x_i)\frac{(t_2-t_1)}{n} = \int_{t_1}^{t_2} v(t)\mathrm{d}t \tag{3-42}$$

在数学上，公式（3-42）被称为积分，记为$l(t) = \int v(t)\mathrm{d}t$。因此可以看到导数和积分就像加减法一样，可以相互推导，互为逆运算。

数学上的导数计算其实就是求函数的极限，而后者通常是比较复杂的。因此数学家们提供了简单函数的导数，以及将复杂函数导数分解为简单函数导数的计算法则。下面将列举导数计算的四则运算法则以及常见简单函数的导数。其中，f和g都是可导函数。

$$(f \pm g)' = f' \pm g'$$
$$(fg)' = f'g + fg'$$
$$\left(f/g\right)' = (f'g - fg')/g^2$$

$$c' = 0;\ (x^n)' = nx^{n-1}$$
$$\sin(x)' = \cos(x);\ \cos(x)' = -\sin(x)$$
$$(\mathrm{e}^x)' = \mathrm{e}^x;\ \ln(x)' = \frac{1}{x} \tag{3-43}$$

3.3.2 极限：变化的终点

在导数的定义中，我们使用了极限这个数学概念。在研究函数微分时，我们又引入了另一个概念：无穷小量$\mathrm{d}t$。这两个概念是数学上非常深刻的结论，在历史上，还曾引起了所谓的第二次数学危机，在此做一个简单的介绍。

当牛顿和莱布尼兹[19]各自独立发明微积分时，他们都使用如图 3-11 所示的推导方法计算函数的导数（以函数x^2为例）。在计算过程中，无穷小量Δx时而不等于 0，时而又等于 0。这就是著名的贝克莱悖论：“幽灵般的无穷小”。

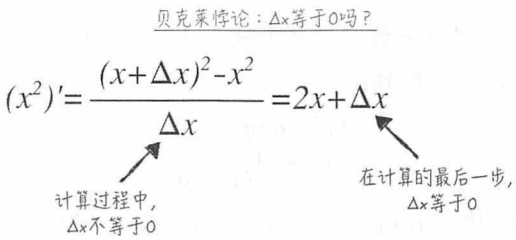


图 3-11

[19] 戈特弗里德·威廉·莱布尼茨（Gottfried Wilhelm Leibniz），德国数学家、哲学家。他与牛顿谁先发明微积分的争论是数学界至今最大的公案。事实上，莱布尼茨对微积分的纪录方法更为简洁明了，现代微积分的数学符号大多起源于他。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

为了解决这个危机，以柯西[20]为首的数学家建立了严格的实数极限理论，将无穷小量理解为一个过程而非一个确定的量。具体地，函数$f(x)$在点x_0的导数$f'(x_0)$定义为：对于任意一个$\varepsilon > 0$，都存在一个$\delta > 0$，使得任何$x \in (x_0 - \delta, x_0 + \delta)$且$x \neq x_0$，都能使公式（3-44）成立（这段表述在数学上是非常著名的$\varepsilon - \delta$语言，相信数学背景较强的读者一定对它非常熟悉）。

$$\left| [f(x) - f(x_0)] / (x - x_0) - f'(x_0) \right| < \varepsilon \tag{3-44}$$

类似地，微分公式$\mathrm{d}f(x) = f'(x)\mathrm{d}x$也表示这样一个动态过程，而非通常意义上的静态相等。

3.3.3 复合函数：链式法则

在公式（3-43）中列举了一些常用函数的导数，但这在现实生活中是不够用的，有很多常用的函数都没有被包括在内。比如多项式函数$f(x) = (x^2 + 1)^2$，它并没有在里面。当然，我们可以根据乘法分配律，将函数$f(x)$改写成最标准的多项式函数，即$f(x) = x^4 + 2x^2 + 1$，然后再根据导数的加法公式，求得$f(x)$的导数：$f'(x) = 4x^3 + 4x = 4x(x^2 + 1)$。但这样计算的效率不高，而且并不是所有的复杂函数都可以如上面那样通过四则运算被分解为简单函数。因此下面将介绍另一种简化复杂函数求导运算的方法：**链式法则**。

事实上，多项式函数$f(x)$可以被看为两个简单函数的复合。不妨设$g(x) = x^2 + 1$，$h(x) = x^2$，则$f(x) = h(g(x))$。通常$f(x)$被称为**复合函数**，记为$f(x) = h \circ g$。

对于复合函数，数学上可以证明如下的等式。

$$(h \circ g)' = (h' \circ g)\ g' \tag{3-45}$$

根据公式（3-45），又因为$h' = 2x$，所以$h' \circ g = 2g(x) = 2(x^2 + 1)$。由此可以很容易得到$f'(x) = 4x(x^2 + 1)$ 。

根据链式法则，可以得到**反函数**的导数公式。不妨设g是f的反函数，即$f(g(x)) = x$。对这个等式的两边求导数，可以得到：

$$\begin{aligned} (f' \circ g)g' &= 1 \\ g' &= 1/f' \circ g \end{aligned} \tag{3-46}$$

比如$f(x) = x^3$的反函数为$g(x) = x^{\frac{1}{3}}$，则根据公式（3-43）以及$f'(x) = 3x^2$。由此可以得到$g'(x) = 1/3(x^{1/3})^2 = 1/3x^{2/3}$。

---

[20] 奥古斯丁·路易·柯西（Augustin Louis Cauchy），法国数学家。他建立了一系列严格的微积分准则，使后者摆脱了贝克莱悖论的困扰。柯西毕业并任教于法国辉煌灿烂的巴黎综合理工学院（Ecole Polytechnique），大学基础课程《高等数学》里的绝大部分理论都源于此学校。

仅供非商业用途或交流学习使用

根据链式法则，可以得到反三角函数的导数：

$$[\arcsin(x)]' = {}^{1}/_{\sqrt{1-x^2}};\ [\arccos(x)]' = {}^{-1}/_{\sqrt{1-x^2}}$$
$$[\arctan(x)]' = {}^{1}/_{1+x^2};\ [\operatorname{arccot}(x)]' = {}^{-1}/_{1+x^2} \tag{3-47}$$

3.3.4 多元函数：偏导数

之前讨论的函数都是单变量函数，即一元函数，但在实际生产中，我们常面对的是多变量函数，即多元函数。因为在数据建模实践中，通常会使用多个因变量，这对应着数学上的多元函数。

对于多元函数，我们使用偏导数来研究函数的局部变化情况。偏导数的基础就是上面介绍的导数，它的定义思路为，选择多元函数中的一个变量作为变量，其他变量被当作常数。然后按照一元函数导数的定义，计算相应的偏导数。比如多元函数$f(x,y) = xy + xy^2$。当对变量x求偏导时，y就被当作常数，反之亦然。由此可以得到：

$$\frac{\partial f}{\partial x} = y + y^2;\ \frac{\partial f}{\partial y} = x + 2xy \tag{3-48}$$

显然，上面介绍的导数计算法则对偏导数同样适用。其中加减乘除法则都比较简单，就不再赘述。而多元复合函数的链式法则比较复杂，接下来着重介绍一下。假设$f(u,v)$是一个二元函数，其中，$u = g(x,y), v = h(x,y)$。数学上可以证明如下等式：

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial x}$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial y} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial y} \tag{3-49}$$

与一元函数的微分类似，在一定条件下[21]，多元函数$f(x_1, x_2, \ldots, x_n)$的微分和偏导数存在如下关系。

$$\mathrm{d}f = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\mathrm{d}x_i \tag{3-50}$$

事实上，f的全部一阶偏导数$(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})$组成了线性空间中的一个向量。这个向量被称为函数的**梯度**，记为∇f。梯度在模型的工程实现上发挥了巨大的作用，具体细节请参考第 3.3.5 节和第 6 章。

3.3.5 极值与最值：最优选择

上面的章节利用大量篇幅讨论了函数的导数和偏导数，那它们的用途是什么呢？答

[21] 当多元函数f在给定点某邻域内的各个偏导数存在且偏导函数在该点都连续，则此函数在该点可微，这时公式（3-50）成立。当然在数据科学领域，处理的函数几乎都是可微函数。

仅供非商业用途或交流学习使用

案是求函数的**极值**（局部最小最大值）和**最值**（全局最小最大值）。其中求函数的最值常被称为**最优化问题**，是数据科学工程实现的核心问题。为了表述简便，先考虑一元可微函数$f(x)$，此函数在x_0处取得最小值。那么根据导数的定义，可以得到$f'(x_0) = \lim_{x \to x_0}[f(x) - f(x_0)]/(x - x_0)$。由于$f(x_0)$是函数的最小值，因此当$x > 0$时，

$$[f(x) - f(x_0)]/(x - x_0) \geqslant 0 \tag{3-51}$$

而当$x < 0$时，

$$[f(x) - f(x_0)]/(x - x_0) \leqslant 0 \tag{3-52}$$

综合公式（3-51）和公式（3-52），可以得到[22]$f'(x_0) = 0$。对于最大值，可以证明有相同的结论。拓展到多元可微函数$g(X)$（其中X为向量），假设它在X_0处取得最值，则它的梯度在此点等于0，即$\nabla g(X_0) = 0$。

当然，函数在某点的导数或梯度等于0，并不能保证它在此点一定取得最值，可能只是函数的**极值点**或**鞍点**（saddle point）等，如图3-12所示。用学术的话来表述，导数或梯度等于0是函数取得最值的必要非充分条件。

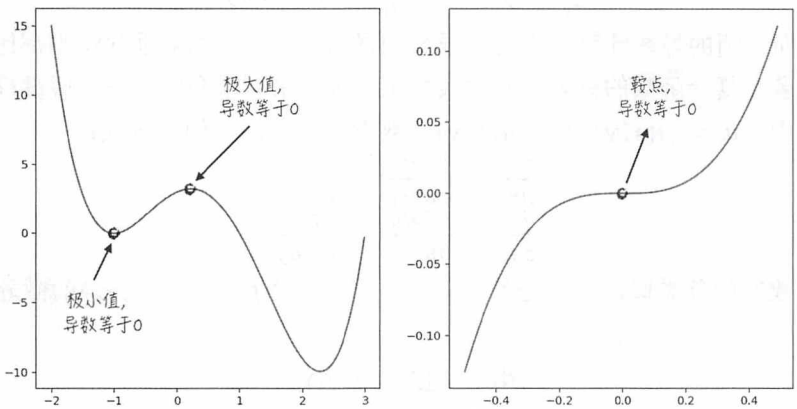


图3-12

尽管不是充要条件，但导数或梯度等于0这个条件提供了筛选可能最值点的方法。在实际中，我们常常使用这个条件来得到备选最值集合，再通过其他方法来最终得到函数的最值。具体细节请参考第6章。

[22] 数学上可以证明，当函数$h(x)$在给定点x_0附近（除去x_0）恒大于等于0时，若极限$\lim_{x \to x_0} h(x)$存在，则一定有$\lim_{x \to x_0} h(x) \geqslant 0$。因此公式（3-49）表示$f'(x_0) \geqslant 0$，而公式（3-50）表示$f'(x_0) \leqslant 0$，因此$f'(x_0) = 0$。

3.4 本章小结

本章简单罗列了数据科学领域常用的数学知识，为之后的模型做理论铺垫。正如本章开头所写的，本章的目的是让读者熟悉后面章节中常遇到的数学概念和数学符号。因此，对于数学定理的证明，往往只做了简单说明，有些定理的严格表述也没有完全展开。还有一些比较复杂的数学工具，比如凸优化（convex optimization），限于篇幅也没有介绍。当然，如果将这些内容全部写完，估计本书就会变成好几本“翻开第一页就不想看的数学书”[23]。

对数学细节感兴趣的读者，可参考其他专业的数学书籍。比如 Kaare Brandt Petersen 和 Michael Syskind Pedersen 编写的 *The Matrix Cookbook*、Eric Lehman 编写的 *Mathematics for Computer Science*、Gilbert Strang 编写的 *Calculus* 以及 Stephen Boyd 和 Lieven Vandenberghe 编写的 *Convex Optimization*。

[23] 这是物理学家杨振宁先生对数学书的评论，原文为“数学书有两种：一种是看了第一章就看不下去的，一种是看了第一页就看不下去的”。但作者完全不同意这个观点。

第4章

线性回归：模型之母

道生一，一生二，二生三，三生万物。

——老子

4.1 一个简单的例子

4.2 上手实践：模型实现

4.3 模型陷阱

4.4 模型持久化

4.5 本章小结

线性回归模型看上去很简单，简单到让不少人觉得它并没有什么研究和使用的价值。其实并不是这样的，线性回归可以说是最重要的数学模型之一，其他很多模型都建立在它的基础之上。为了更好地理解这一点，让我们先来看一个有关数学家的笑话：

一天，数学家觉得自己已受够了数学，于是他跑到消防队去宣布他想当消防员。

消防队长说："您看上去不错，可是我得先给您一个测试。" 消防队长带数学家到消防队后院小巷，巷子里有一个货栈、一只消防栓和一卷软管。

消防队长问："假设货栈起火，您怎么办？"

数学家回答："我把消防栓接到软管上，打开水龙，把火浇灭。" 消防队长说："完全正确！最后一个问题：假设您走进小巷，而货栈没有起火，您怎么办？"

数学家疑惑地思索了半天，终于答道："我就把货栈点着。" 消防队长大叫起来："什么？太可怕了！您为什么要把货栈点着？"

数学家回答："这样我就把问题化简为一个我已经解决过的问题了。"

搭建模型的思路和笑话中数学家的思路一样。当遇到一个新问题时，总是考虑通过某种数学上的变换，将未知问题转化为已知模型能解决的问题。所以任何一个复杂模型，一层层拨开来， 里面可能藏着好多个线性回归模型。因此，线性回归模型很有研究的必要。深入了解它的模型细节能帮助我们理解其他模型，进而指引我们根据实际场景搭建有效的模型。

再来说说线性回归的使用价值。读者可能会觉得，线性回归太过简单，应该没人会使用它来解决实际问题吧。但事实上，在经济学领域，有很多模型都是线性回归模型。比如宏观经济学的基石之一——IS-LM 模型[1]。这些模型的结果深刻地影响着政府的经济决策，进而影响着我们生活的方方面面。因此可见，线性回归并不缺乏实用例子。

而且搭建模型并不是要完全模拟现实世界，而是一个不断做近似的过程。这就是为什么数据科学家们常说"所有模型都是错的"。当然这句话还有后半句"但是，其中一些是有用的"。一个"有用"的模型能过滤掉数据中那些不重要的细枝末节，抓住其中主要的内在关系，从而帮助我们更好地理解和解释数据。而在很多情况下，线性模型就是这么一个符合要

---

[1] IS-LM 模型由两个线性模型组成，一个是 IS 模型，即投资-储蓄模型（Investment-Saving），它研究的是国民储蓄和市场利率之间的关系；另一个是 LM 模型，即流动性偏好-货币供给模型（Liquidity Preference-Money Supply），它研究的是国民收入与货币需求量之间的关系。这两个模型都只涉及两个变量：Y表示国民收入，可以简单地理解为日常生活中常听到的 GDP；r表示利率。

- 对于投资储蓄行为，假设在短期内，国民收入Y增加，这意味着人们可以用于储蓄的资金增加，市场上的利率会下降。因此在 IS 模型中，$r = -aY + b$，其中，$a > 0$。
- 对于货币供给，假设在短期内，央行的货币供应量是不变的，而且国民收入Y增加。由于收入增加，人们对货币的需求量是增加的，而货币供应量并没有增加，这会抬高市场上的利率。因此在 LM 模型中，$r = cY + d$，其中$c > 0$。

将这两个结合起来，就可以得到市场上的利率以及对应的国民收入。更进一步，根据想要达到的利率和国民收入可以倒推出央行的货币供应量。IS-LM 模型充分说明："模型不在复杂，有用则灵"。

求的“有用”模型，而且它还简洁、高效、容易理解。既然线性模型已经够用了，那又何必再费劲去构造那些难以理解，同时又极易引入其他新问题的复杂模型呢[2]？所以线性回归模型在现实世界里有很强的使用价值。

说了这么多有关线性模型的好处和重要性，下面来深入讨论这个模型的细节。

4.1 一个简单的例子

让我们用一个简单的例子来引出线性回归模型。为了更形象地描述，假设我们有一个制作手工玩偶的朋友小潘，小潘想分析一下他制作玩偶的数量和成本之间的关系。于是他一边生产玩偶，一边记录了如图 4-1 所示的数据。

生产记事本

日期	玩偶个数	成本	第几天
04/01	10	7.7	1
04/02	10	9.87	2
04/03	11	10.87	3
04/04	12	12.18	4
04/05	13	11.43	5
04/06	14	13.36	6
04/07	15	15.15	7
04/08	16	16.73	8
04/09	17	17.4	9
...	...	...	...

图 4-1

小潘仔细研究了一下他所记录的数据，似乎玩偶个数和成本是线性关系。这也很符合小潘的预期，为了更加直观地了解数据，验证他的猜测，他决定将数据可视化。他把上面的数据点表示在一个直角坐标系里，如图 4-2 所示。

---

[2] 这里涉及一个很重要的哲学准则，它叫作奥卡姆剃刀（Occam's Razor），也称为简约之法则。这个准则在机器学习领域应用得很广，其主要含义可概括为“如无必要，切勿假定繁多”。也就是说，如果关于同一个问题有许多种理论，每一种都能做出同样准确的预言，那么应该挑选其中使用假定最少的。尽管越复杂的方法通常能做出越好的预言，但是在不考虑预言能力（即结果大致相同）的情况下，假设越少越好。在建模的过程中，我们大多遵循这一原则。

奥卡姆剃刀最成功的案例来自于天文学。在日心说刚刚诞生时，太阳、月亮和其他太阳系行星的运动既可以用地心说来解释，也可以用日心说来解释。在当时看来，两种假说都同样有效，然而，日心说只需要 7 个基本假设，地心说却需要很多的假设。那么基于奥卡姆剃刀原则，应该选择日心说。

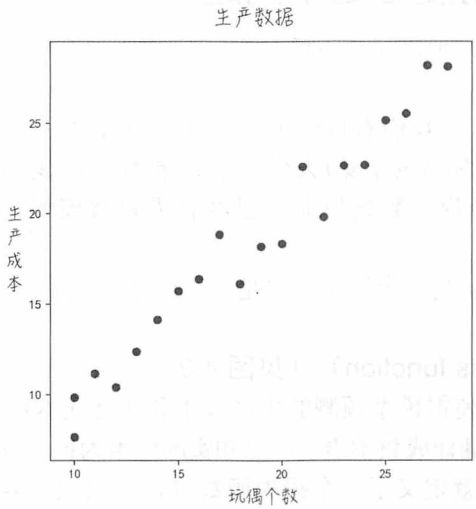


图 4-2

根据图像，小潘发现，生产成本和生产个数并不呈一条严格的直线，似乎是在沿着某条直线上下随机地波动。

其实上面展示的这些数据，是由“自然之力”按照下面这个数学公式来产生的[3]。

$$y_i = x_i + \varepsilon_i \tag{4-1}$$

- x_i是某一天小潘制作的玩偶个数，y_i是对应那天的生产成本。这个数学式子告诉我们，小潘制作玩偶的平均成本是 1。
- 其中$\{\varepsilon_i\}$是一个随机变量，它服从期望为 0，方差为 1 的正态分布。它表示在小潘生产玩偶时，一些随机产生或随机节约的成本。比如制作失败的玩偶，这时ε_i为正；又比如制作过程中刚好发现有可用的旧布料，这时候ε_i为负。
- ε所代表的随机成本和制作玩偶的个数是相互独立的。

但是，小潘并不是控制公式的“自然之力”。他所看到的只是一堆数据：玩偶个数$\{x_i\}$和对应的生产成本$\{y_i\}$。而他看不到的公式（4-1）正是他想要的结果。

小潘并不知道如何对数据建模分析，所以他将问题提给了他的两个好朋友：一位是从事机器学习工作的小陈，另一位是进行统计工作的小郭。我们来看看这两位分别打算怎么解决这个问题呢？

---

[3] 这个公式其实并不严谨。因为正态分布的值域是整个实数，所以根据公式 1，y_i的取值可能为负数。这与生产成本必然大于 0 相矛盾。数学上完全严格的写法应为$y_i = \max(x_i + \varepsilon_i, 0)$。但是在书中这个场景里，$x_i + \varepsilon_i < 0$的概率非常小，几乎可以忽略，所以模型可近似为公式（4-1）。

4.1.1 从机器学习的角度看这个问题

小陈听完小潘的描述后，他做了如下的分析。

1．确定场景类型

（1）在现有的数据集里，我们有玩偶的生产个数（记为x_i）和生产成本（记为y_i）。其中i表示第i天的数据，比如x_i表示第i天生产的玩偶数。而我们需要通过生产个数的信息去预测生产成本。也就是说在数据里面，已经有需要被模型预测的量：生产成本，所以这是一个监督式学习。

（2）需要被预测的成本y_i是一个数量。它是一个连续变化的量，而并非表示类别的离散量，所以这是一个回归问题。

2．定义损失函数（loss function）（见图 4-3）

（1）搭建模型的目标是使得模型预测的生产成本和实际成本接近。

（2）我们把上面这句话翻译成数学语言：已知实际成本为y_i，假设模型预测的成本为$\hat{y}_i$。那么预测值和真实值的差距就定义了一个损失函数LL，如公式（4-2）所示。而搭建模型的目标就是使得损失函数达到最小值。

$$LL = \sum_i |y_i - \hat{y}_i| \tag{4-2}$$

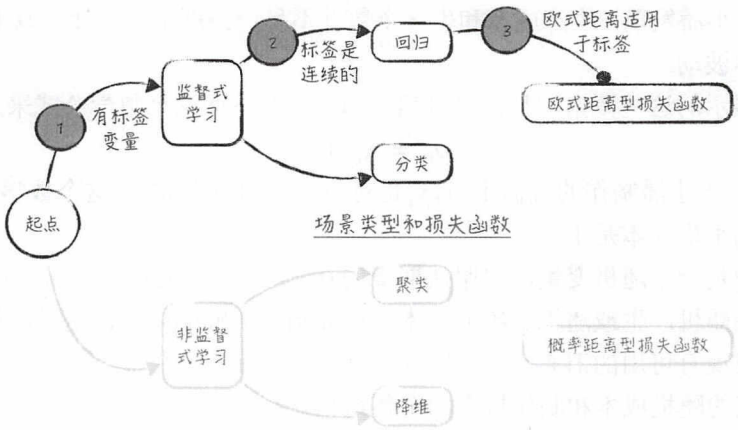


图 4-3

（3）公式（4-2）并不是一个处处可导的函数，数学上处理起来比较麻烦。因此，我们重新定义一个数学上容易处理的损失函数[4]，如公式（4-3）所示（其实就是真实值与预测值之间的欧式距离平方和）。模型参数的估计就依赖于这个损失函数：

---

[4] 对于线性回归模型，公式（4-3）所定义的损失函数除了在数学上更易处理外，它还隐含地满足了数据的一个假设，即随机扰动项服从正态分布。

事实上，公式（4-2）和公式（4-3）对应着不同的线性回归模型。公式（4-2）对应着 Least Absolute Deviations Regression，而公式（4-3）对应着 Least Squares Regression。

$$L = \sum_i (y_i - \hat{y}_i)^2 \tag{4-3}$$

3．提取特征（见图 4-4）

（1）数据是小潘自己收集记录的。经过检查，里面并没有记错或者特别异常的数据。换句话说，数据是“干净”的，可以直接使用。

（2）现有的数据里，只有一个原始特征$\boldsymbol{X} = \{x_i\}$，它表示个数。在这个变量上面的加减乘除是有明确含义的。也就是说变量本身的数学运算是有意义的，所以$\boldsymbol{X}$可以直接在模型里面使用。

（3）当然也可以对$\boldsymbol{X}$做某种数学变换，得到一个新的特征。比如对它做平方运算，得到新的特征$\boldsymbol{X}^2$。这些新提取的特征也可以被应用到模型里。但对小潘的问题，我们先只用原始特征建模。如果效果不好，则再考虑提取新的特征。

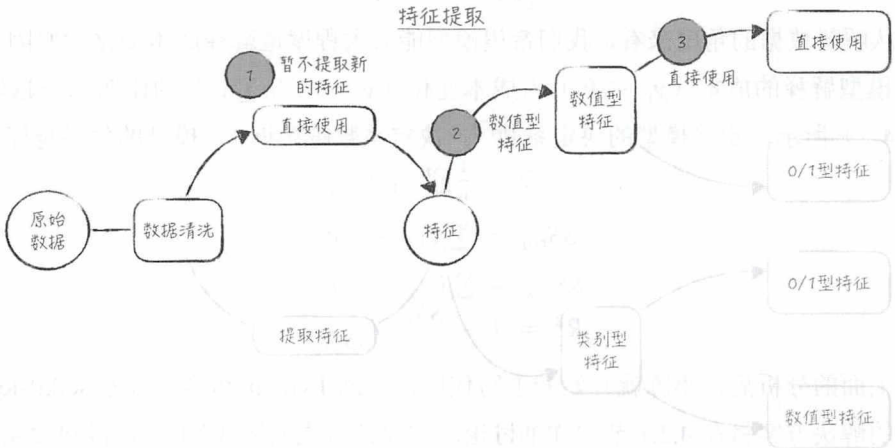


图 4-4

4．确定模型形式并估计参数

（1）根据小潘的分析，x_i和y_i之间是线性关系，因此可以直接使用线性模型，不需要考虑非线性问题到线性问题的转化了。

（2）模型的定义如公式（4-4），其中，a表示生产一个玩偶的变动成本，b表示生产的固定成本[5]。

---

[5] 如果用模型的语言来描述，公式（4-4）中的b称为截距，它是线性回归模型中非常重要的参数。在加入这个参数后，模型才能在数据平移和数据单位转换时保持稳定。具体来讲：

- 当发生数据平移时，比如小潘发现，他原来统计的成本中少算了 1 元，现在需要加上，即y_i将变为$y_i + 1$。参数a（平均成本）的估计值不变。
- 当发生数据单位转换时，比如小潘将成本的单位由元改为了百元，即y_i将变成$y_i/100$。参数a将成比例的放大或缩小，比如上面的例子里，a将变成$a/100$。

上面的两点性质对线性回归，特别是多元线性回归十分重要。因为当模型有多个自变量时，它们的单位常常是不一致的。如果没有加入b这个参数，这两点性质便不再成立。所以在搭建线性回归模型时，通常会在模型中加入截距。除非给定的数据集里已经包含了一个常量自变量。

$$\hat{y}_i = ax_i + b \tag{4-4}$$

（3）如上面提到的那样，参数(a, b)的估计值$(\hat{a}, \hat{b})$将使得损失函数L达到最小值，如公式（4-5）所示。

$$(\hat{a}, \hat{b}) = \mathrm{argmin}_{a,b} \sum_i (y_i - ax_i - b)^2 \tag{4-5}$$

5．评估模型效果

（1）从预测的角度看，我们希望模型的预测成本越接近真实成本越好。所以如公式（4-6）所示，定义线性模型的均方差[6]。均方差越小，模型效果越好。

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}L \tag{4-6}$$

（2）从解释数据的角度来看，我们希望模型能最大程度地解释成本变化的原因。换句话说，未被模型解释的成本（$y_i - \hat{y}_i$）占成本变化（$y_i - 1/n\sum y_i$）的比例越小越好。因此如公式（4-7）所示，定义模型的决定系数[7]。决定系数越接近 1，模型的效果越好。

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

$$SS_{tot} = \sum_i (y_i - \bar{y})^2$$

$$SS_{res} = \sum_i (y_i - \hat{y}_i)^2$$

$$R^2 = 1 - {SS_{res}}/{SS_{tot}} \tag{4-7}$$

做完上面的分析后，小陈就开始着手写代码，借助 Python 和第三方库 scikit-learn 解决问题。他的解决方案将在 4.2.1 节中详细讨论。建议读者先阅读 4.2.1 节，再阅读 4.1.2 节。

---

[6] 均方差的英文名为 Mean Squared Error，通常缩写为 MSE。细心的读者会发现，均方差就是上面定义的损失函数L的平均值。所以均方差也可作为指标，用于判断数据是否应该使用线性模型。

[7] 决定系数的英文名为 coefficient of determination，通常记为R^2，如图 4-5 所示。

使用同正文中一样的符号，通过数学运算，在最小化损失函数时，有$\sum_i y_i = \sum_i \hat{y}_i$。由此可以得到：

$$\sum_i (\hat{y}_i - \bar{y})^2 + \sum_i (y_i - \hat{y}_i)^2 = \sum_i (y_i - \bar{y})^2$$

其中，$SS_{tot} = \sum_i (y_i - \bar{y})^2$是因变量的方差，$SS_{reg} = \sum_i (\hat{y}_i - \bar{y})^2$是已被模型（或者说是自变量）解释的方差。所以在这种情况下，有：

$$R^2 = SS_{reg}/SS_{tot}$$

用学术语言来解释，这个公式说明：决定系数等于因变量的方差中可由自变量解释的比例。因此，可以用这个指标来判断模型的解释力。

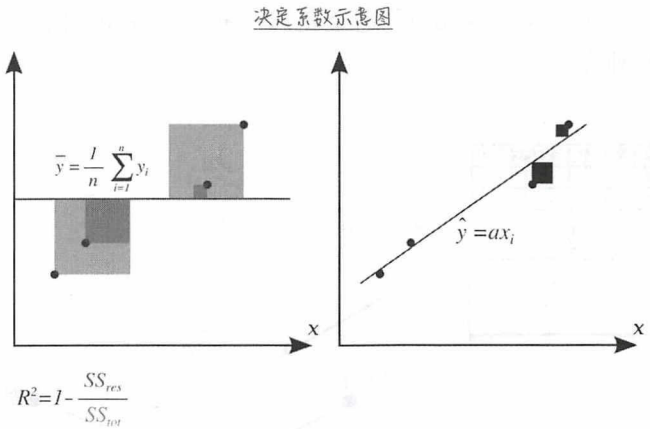


图 4-5[8]

4.1.2 从统计学的角度看这个问题

作为统计工作者的小郭，先从数学角度分析了这个问题：

1．假设条件概率

（1）根据小潘的描述，数据集里有两个变量，一个是自变量[9]玩偶个数（记为x_i），一个是因变量生产成本（记为y_i）。其中i表示第i天的数据，比如x_i表示第i天生产的玩偶数。而且根据小潘前面的分析，这两者之间似乎是线性关系，但又带着一些随机波动。因此可以假设y_i和x_i之间的关系如下：

$$y_i = ax_i + b + \varepsilon_i \tag{4-8}$$

（2）其中，a,b是模型的参数，分别表示生产一个玩偶的变动成本和固定成本；而ε_i被称为噪声项，表示没被已有数据所捕捉到的随机成本。它服从期望为 0，方差为σ^2（σ^2也是模型的参数）的正态分布，记为$\varepsilon_i \sim N(0,\sigma^2)$。这里假设$\{\varepsilon_i\}$之间相互独立，而且$\{\varepsilon_i\}$和$\{x_i\}$之间也是相互独立的，这两点假设非常重要[10]。

（3）从左到右看公式（4-8），如果给定一组参数a,b以及噪声项的方差σ^2。由于x_i表示玩偶个数，是一个确定的量（小潘能控制他生产的玩偶数）。那么y_i就和ε_i一样是一个随机变量，服从期望为$ax_i + b$，方差为σ^2正态分布，即$y_i \sim N(ax_i + b,\sigma^2)$。换句话说，

[8] 图片来源于维基百科。

[9] 细心的读者会注意到，对于数据里的玩偶个数x_i，小陈使用“特征”这个术语来描述它，而小郭使用的术语是“自变量”。这两个术语的含义并无差异，只是“特征”这个术语常用于机器学习领域，而“自变量”这个术语常用于统计学领域。

[10] 如果假设不成立，情况会怎样呢？本书将在第 7 章中详细讨论相关问题。

书籍是人类进步的阶梯

小潘提供的数据y_i，只是$N(ax_i + b, \sigma^2)$这个正态分布的一个观测值，如图 4-6 所示，而且$\{y_i\}$之间也是相互独立的。

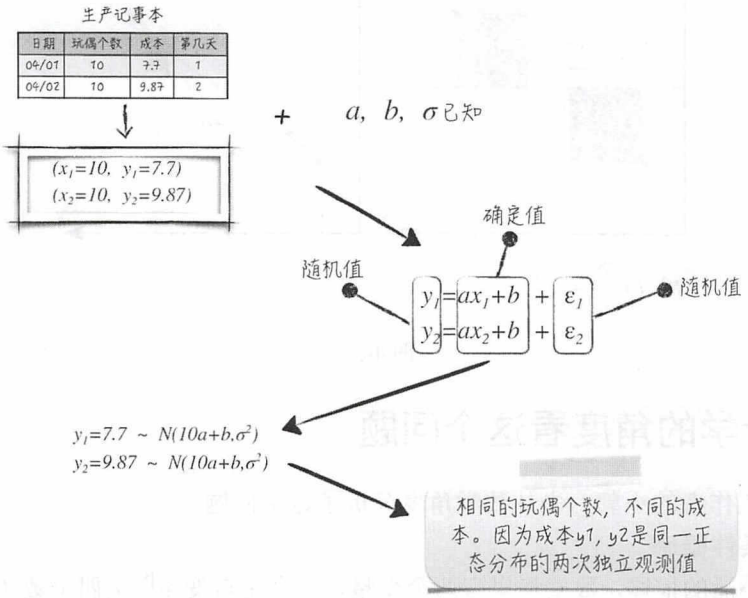


图 4-6

（4）把上面的第（3）点翻译成数学语言就是：y_i在已知a, b, x_i, σ时的条件概率是$N(ax_i + b, \sigma^2)$，如公式（4-9）所示。

$$P(y_i|a, b, x_i, \sigma^2) \sim N(ax_i + b, \sigma^2) \tag{4-9}$$

2. 估计参数

（1）根据上面的分析，$\{y_i\}$之间也是相互独立的。所以得到$\{y_i\}$出现的联合概率如公式（4-10）所示。在学术上，这个概率被称为模型的似然函数（likelihood function），通常也被记为L[11]。

$$P(Y \mid a, b, X, \sigma^2) = \prod P(y_i \mid a, b, x_i, \sigma^2)$$
$$\ln P(\boldsymbol{Y} \mid a, b, \boldsymbol{X}, \sigma^2) = -0.5n\ln(2\pi\sigma^2) - (1/2\sigma^2)\sum_i(y_i - ax_i - b)^2 \tag{4-10}$$

（2）对于不同的模型参数，$\{y_i\}$出现的概率（即参数的似然函数）并不相同。这个概率当然是越大越好，所以使这个概率最大的参数将是参数估计的最佳选择。此方法也被称

[11] 在学术上，通常用θ表示模型中的所有参数，比如正文中的a, b, σ^2；用D表示数据。则参数的似然函数记为$L(\theta \mid D) = P(D \mid \theta)$。值得注意的是，等式右边的$P(D \mid \theta)$表示条件概率，而等式的左边$L(\theta \mid D)$，虽然形式与右边相似，但不表示条件概率。这一点通常会让初学者感到疑惑。

书籍是人类进步的阶梯

为最大似然估计法（Maximum Likelihood Estimation，MLE）。根据公式（4-10），参数(a,b)的估计值$(\hat{a},\hat{b})$如下[12]：

$$(\hat{a},\hat{b}) = \mathrm{argmax}_{a,b} P(\boldsymbol{Y} \mid a,b,\boldsymbol{X},\sigma^2) = \mathrm{argmin}_{a,b} \sum_i (y_i - ax_i - b)^2 \tag{4-11}$$

（3）同理，可以得到参数σ^2的估计值$\widehat{\sigma^2}$，如公式（4-12）所示。

$$\widehat{\sigma^2} = \mathrm{argmax}_{a,b} P(\boldsymbol{Y} \mid a,b,\boldsymbol{X},\sigma^2) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 / n$$

$$\hat{y}_i = \hat{a}x_i + \hat{b} \tag{4-12}$$

3．推导参数的分布

（1）其实上面得到的参数估计值$(\hat{a},\hat{b},\widehat{\sigma^2})$都是随机变量。具体的推导过程有些繁琐，限于篇幅，这里略去数学细节。仅以$\hat{a}$为例，用一个不太严谨的数学推导来说明这个问题[13]。假设数据集里只有两对数据$(x_k,y_k),(x_l,y_l)$，且$x_k \neq x_l$。这时我们可以通过解公式（4-13）中的方程组来得到$\hat{a}$的表达式。公式（4-13）的右半部分表示$\hat{a}$是一个随机变量，而且服从一个以参数真实值a为期望的正态分布[14]。

$$\begin{cases} y_k = \hat{a}x_k + \hat{b} \\ y_l = \hat{a}x_l + \hat{b} \end{cases} \Rightarrow \hat{a} = \frac{y_k - y_l}{x_k - x_l} = \frac{a(x_k - x_l) + \varepsilon_k - \varepsilon_l}{x_k - x_l} = a + \frac{\varepsilon_k - \varepsilon_l}{x_k - x_l} \tag{4-13}$$

（2）通过更加细致的数学运算可以得到：

$$\begin{gathered} \hat{b} \sim N\left(b, \sigma^2/n\right) \\ \hat{a} \sim N\left(a, \sigma^2 / \sum_i (x_i - \bar{x})^2\right) \\ \widehat{\sigma^2} \sim \chi^2_{n-2} \frac{\sigma^2}{n} \end{gathered} \tag{4-14}$$

（3）既然参数估计值都是随机变量，那么我们更需要关心的是这些估计值所服从的概率分布，而不仅仅是根据公式（4-11）和公式（4-12）得到的数值。因为这些数值只是对应分布的一次观测值，它们并不总是等于真实参数，而是严重依赖于估计参数时所使用的数据。比如针对小潘提供的数据集，只使用前 3 天数据估计出来的参数和使用 4～6 天数据估计出

---

[12] 细心的读者会发现公式（4-5）和公式（4-11）是一样的。小陈和小郭的假设和建模出发点完全不同，但是殊途同归，得到的结果却是一致的。

[13] 严谨的数学证明如下：用矩阵来表示线性回归模型。令

$$\boldsymbol{Y} = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix}, \boldsymbol{X} = \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \cdots & \\ x_n & 1 \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} a \\ b \end{pmatrix}, \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \cdots \\ \varepsilon_n \end{pmatrix}$$

则模型可表达为$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$。而参数$\boldsymbol{\beta}$的估计为$\hat{\boldsymbol{\beta}} = \mathrm{argmin}_{\beta}(\boldsymbol{Y} - \boldsymbol{X\beta})^2$。进一步的数学运算可以得到：

$$\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X\beta} + (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\varepsilon}$$

这个式子表示，模型参数的估计值是随机变量，而且它们服从一个期望为参数真实值的正态分布。

[14] 如果两个服从正态分布的随机变量相互独立，则它们的线性组合依然是正态分布。

书籍是人类进步的阶梯

来的参数就不一样，如图 4-7 所示。

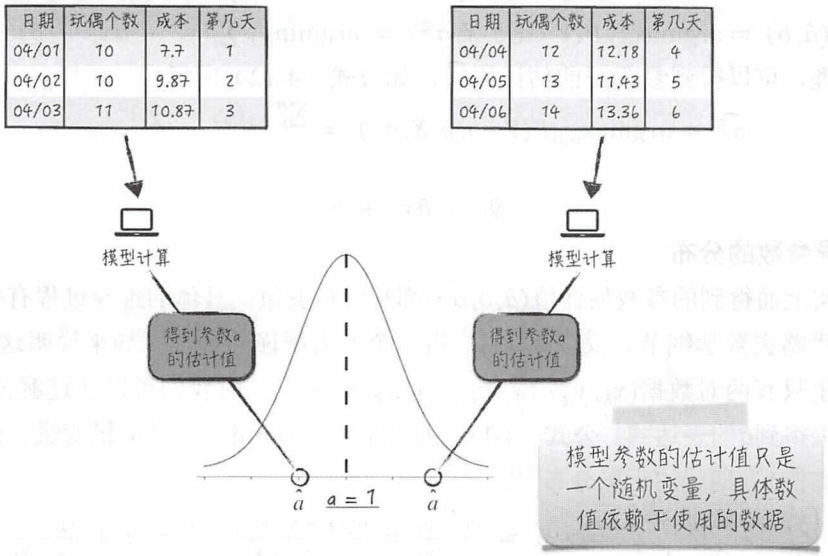


图 4-7

（4）公式（4-14）告诉我们，参数估计值的方差随着数据量的增大而减少。换句话说，数据量越大，模型估计的参数就越接近真实值。这也是大数据的价值之一：数据量越大，模型预测的效果就越好，如图 4-8 所示。

4. 假设检验与置信区间

（1）参数的概率分布可以透露许多很有用的信息。比如在 95%的情况下，参数a的真实值（生产一个玩偶的变动成本）会落在一个怎样的区间里？在学术上它被称为参数a的 95%置信区间。类似地，也可以计算模型预测结果的置信区间，即对于被预测对象，真实值的大致范围是怎样。这一点非常重要，因为模型几乎不可能准确地预测真实值。知道真实值的概率分布情况，能使我们更有信心地使用模型结果。

（2）又比如在 1%犯错的概率下，我们能不能拒绝参数b的真实值其实等于 0 这个假设？在学术上它被称为参数b的 99%显著性假设检验。对于这个假设检验，换个思路更通俗一点理解：参数b的真实值等于 0 的概率是否小于 1%？这可以帮助我们更好地理解数据之间的关系，比如小潘生产玩偶时，固定成本（参数b）是否真实存在？或者模型估计的固定成本$\hat{b}$只是由于模型搭建得不准确而导致的“错误”结论？

小郭分析完问题之后，就着手利用 Python 和第三方库 Statsmodels 编写自己的解决方案。他给出的方案将在 4.2.2 节中详细展示。

书籍是人类进步的阶梯

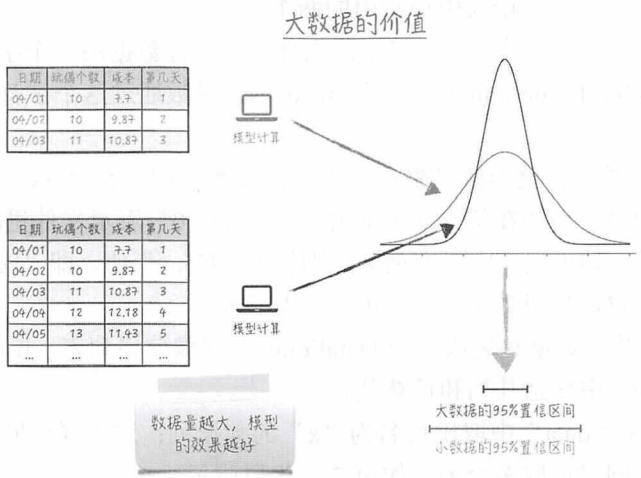


图 4-8

4.2 上手实践：模型实现

在讨论模型代码之前，我们看看如何利用 pandas 方便地读取数据。在阅读的同时，读者可以打开 Python shell 或者 IPython，跟随本书一起练习。

程序清单 4-1 读取数据——pandas

```
1  | >>> import pandas as pd
2  | >>> # 确保 path 等于数据路径，读取数据
3  | >>> data = pd.read_csv(path)
4  | >>> data = pd.read_csv(path, delimiter=",", header=0)
5  | >>> # 选择列
6  | >>> data["x"]
7  | >>> data[["x", "y"]]
8  | >>> data.x
9  | >>> # 选择行
10 | >>> data[:10]
11 | >>> data[10:]
12 | >>> data[2:3]
```

（1）如程序清单 4-1 中的第 1 行代码所示，导入第三方库 pandas，并给它取别名“pd”。这是约定俗成的，目的是简略好记。在后面使用 pandas 时，用字符“pd”代替，减少字符输入量。这在 Python 里面特别常见，比如在第 2 章中出现过的用“np”代替 numpy 等。

（2）pandas 库提供了读取 csv 文件的函数 read_csv。通常情况下，只需传入数据所在路径即可，如第 3 行代码所示。也可以通过它提供的参数来定制读取数据的格式，如第 4 行代

书籍是人类进步的阶梯

书籍是人类进步的阶梯

码所示，限于篇幅，这里只讨论其中最常用的两个。

- delimiter：参数类型为字符型。在 csv 文件中，一行数据用一个字符串来储存。列数据与列数据之间用分隔符（delimiter）隔开。read_csv 函数通过这个字符，将原始的字符串转化了一列一列的数据。
- header：参数类型为整形或者整形列表。该参数表明，在“csv”文件里，第几行数据将被作为数据的列名。比如在第 4 行代码中，“header=0”表示文件里的第 1 行是数据的列名，而非数据本身。如果文件中没有列名，则传入“None”值，即“header=None”，这时候程序会自动根据位置，生成数字列名：0，1，2，…

（3）read_csv 函数将数据封装成一个 DataFrame。就像使用列表（list）一样，你可以很方便地选择 DataFrame 中任意几列和任意几行。

- 第 6 行代码从“data”中取出列名为“x”的数据列；这个操作同样可以由第 8 行代码实现。我们也可以同时选取多个列，如第 7 行代码所示。
- 对于行的选取，则完全跟 Python 的列表一致，如第 10～12 行代码所示。

4.2.1 机器学习代码实现

本节我们将讨论小陈提供的解决方案[15]。在讨论代码之前，我们先来看看他的模型效果，如图 4-9 所示。

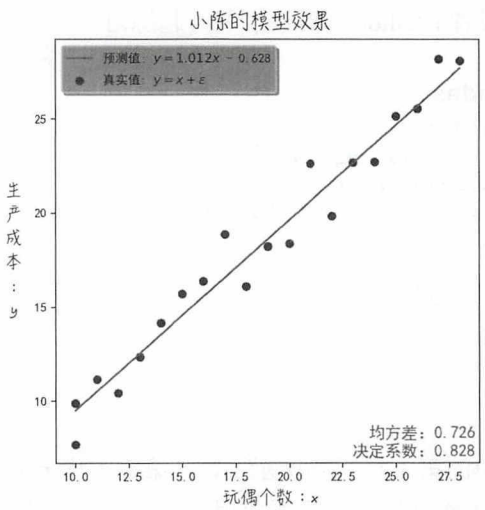


图 4-9

[15] 完整的代码请参考随书配套的代码/ch04-linear/simple_example/linear_ml.py。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

- 小陈估计模型是$\hat{y}_i = 1.012x - 0.628$。也就是说他估计，玩偶的平均成本为 1.012，生产的固定成本为−0.628。估计的平均成本与真实的生产成本 1 相差并不大。但是估计的固定成本是一个负数，这有些违背常识。
- 模型的均方差被估算为 0.726。它表示，如果用这个模型去估计生产总成本，平均的误差为 0.85（$\sqrt[2]{0.726} \approx 0.85$）。
- 模型的决定系数等于 0.828。这表示大约 83%的成本变化可以由模型解释，即 83%的成本是由玩偶制作过程产生的。这部分成本是可控的，与生产个数是严格的线性关系。而剩下的 17%是暂时未知的随机成本。

下面我们将讨论小陈的模型实现。

从整体结构来看，小陈的解决方案分为 4 步，如程序清单 4-2 所示。

（1）将数据划分为训练数据集和测试数据集。这一步是为了解决过度拟合的问题，这部分内容在这里先不做介绍，将在 4.3.1 节再做详细讨论。

（2）利用训练数据集训练模型，估计模型参数。

（3）利用测试数据集评价模型，计算对应的均方差和决定系数。

（4）用图形化的方式，展示模型效果。

程序清单 4-2 线性回归

```
def linearModel(data):
    """
    线性回归模型建模步骤展示

    参数
    ----
    data : DataFrame，建模数据
    """
    features = ["x"]
    labels = ["y"]
    # 划分训练集和测试集
    trainData = data[:15]
    testData = data[15:]
    # 产生并训练模型
    model = trainModel(trainData, features, labels)
    # 评价模型效果
    error, score = evaluateModel(model, testData, features, labels)
    # 图形化模型结果
    visualizeModel(model, data, features, labels, error, score)
```

小陈使用第三方开源算法库 scikit-learn 来搭建和训练他的线性回归模型。就像程序清单 4-3 所展示的那样，整个过程十分简洁。除去说明文档外，其实就两行代码。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

（1）首先创建所需的线性回归模型“linear_model.LinearRegression”，如第 20 行代码所示[16]。

（2）再使用训练数据估计模型参数，如第 22 行代码所示[17]。值得注意的是， model.fit 函数一旦被调用，它将根据传入的数据，估计模型参数并修改对应的对象“model”。所以在第 22 行代码里，虽然没有任何赋值，但“model”在这行代码里已经被修改了。第 22 行代码之后的“model”就是用数据训练好的模型。

程序清单 4-3 训练模型——scikit-learn

```
 1 | from sklearn import linear_model
 2 |
 3 | def trainModel(trainData, features, labels):
 4 |     """
 5 |     利用训练数据，估计模型参数
 6 |
 7 |     参数
 8 |     ----
 9 |     trainData : DataFrame，训练数据集，包含特征和标签
10 |
11 |     features : 特征名列表
12 |
13 |     labels : 标签名列表
14 |
15 |     返回
16 |     ----
17 |     model : LinearRegression，训练好的线性模型
18 |     """
19 |     # 创建一个线性回归模型
20 |     model = linear_model.LinearRegression()
21 |     # 训练模型，估计模型参数
22 |     model.fit(trainData[features], trainData[labels])
23 |     return model
```

小陈评价模型的代码也十分简单，如程序清单 4-4 所示。

（1）对象“model”有一个 predict 函数。它对传入的特征数据使用已经训练好的模型，以得到相应的预测。如第 25 行代码中的“model.predict(testData[features])”。

（2）对象“model”还有一个 score 函数。它将计算模型在传入数据集上的决定系数。如第 27 行代码中的“model.score(testData[features], testData[labels])”。

---

[16] 可以通过“LinearRegression”的参数“fit_intercept”（默认为 True）来决定是否在模型中加入截距，即公式（4-4）中的b。比如代码“linear_model.LinearRegression(fit_intercept=False)”表示创建一个没有截距的线性回归模型。

[17] 参考第 22 行代码，读者可以通过修改训练数据来验证截距项的作用。具体来讲，先分别创建有截距和没有截距的线性回归模型，并在模型的基础上进行数据平移以及数据单位转换，观察模型参数的变化。其中：

- 数据平移对应的代码是“model.fit(trainData[features], trainData[labels] + 1)”；
- 数据单位转换对应的代码是“model.fit(trainData[features], trainData[labels] / 100.0)”。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

程序清单 4-4 评价模型

```
1  import numpy as np
2
3  def evaluateModel(model, testData, features, labels):
4      """
5      计算线性模型的均方差和决定系数
6
7      参数
8      ----
9      model : LinearRegression, 训练完成的线性模型
10
11     testData : DataFrame, 测试数据
12
13     features : list[str], 特征名列表
14
15     labels : list[str], 标签名列表
16
17     返回
18     ----
19     error : np.float64, 均方差
20
21     score : np.float64, 决定系数
22     """
23     # 均方差(The mean squared error), 均方差越小越好
24     error = np.mean(
25         (model.predict(testData[features]) - testData[labels]) ** 2)
26     # 决定系数(Coefficient of determination), 决定系数越接近 1 越好
27     score = model.score(testData[features], testData[labels])
28     return error, score
```

最后，小陈使用第三方库 Matplotlib 将模型结果表现在图上，方便小潘直观理解。虽然可视化对数据科学很重要，特别是与非技术人员交流时，一图胜千言。但这部分的实现比较繁琐和机械化，限于篇幅，这里就不做展示和讨论了，请读者参考随书配套的代码。

4.2.2 统计方法代码实现

本节我们将讨论小郭的解决方案[18]。不同于上一节，这里将着重展示小郭分析数据，修改模型的过程。

小郭首先利用第三方库 Statsmodels 训练事先假定好的线性回归模型，即$y_i = ax_i + b + \varepsilon_i$，如程序清单 4-5 所示。

（1）由于在模型中假设了固定成本的存在，即参数b，所以需要在已给的自变量中加入一个常量变量[19]。如第 15 行代码所示，使用 add_constant 函数来完成这一操作，新加入的

[18] 完整的代码请参考随书配套的代码/ch04-linear/simple_example/linear_stat.py。

[19] 在第三方库 Statsmodels 中，线性回归模型是用矩阵的形式来表示。因此如果要引入参数b，需要在自变量中加入常变量 1。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

常量变量的变量名是“const”。

（2）使用 Statsmodels 搭建并训练模型的代码也很简单，如第 25、26 行代码所示。其中，第 25 行代码表示创建一个线性回归模型，而第 26 行代码表示训练模型，估计模型参数。

（3）对于训练好的模型，可以通过调用 summary 函数来计算模型最常用的统计性质；也可以调用 f_test 来做假设检验。具体的代码可参考第 34～40 行。具体的调用结果将在随后详细讨论。

程序清单 4-5 分析线性回归模型——Statsmodels

```
1  import statsmodels.api as sm
2
3  def linearModel(data):
4      """
5      线性回归统计性质分析步骤展示
6
7      参数
8      ----
9      data : DataFrame，建模数据
10     """
11     features = ["x"]
12     labels = ["y"]
13     Y = data[labels]
14     # 加入常量变量
15     X = sm.add_constant(data[features])
16     # 构建模型
17     re = trainModel(X, Y)
18     # 分析模型效果
19     modelSummary(re)
20
21 def trainModel(X, Y):
22     """
23     训练模型
24     """
25     model = sm.OLS(Y, X)
26     re = model.fit()
27     return re
28
29 def modelSummary(re):
30     """
31     分析线性回归模型的统计性质
32     """
33     # 整体统计分析结果
34     print re.summary()
35     # 用 f test 检验 x 对应的系数 a 是否显著
36     print re.f_test("x=0")
37     # 用 f test 检验常量 b 是否显著
38     print re.f_test("const=0")
39     # 用 f test 检验 a=1, b=0 同时成立的显著性
40     print re.f_test(["x=1", "const=0"])
```

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

接下来，小郭对第 1 步中的模型结果进行分析。调用程序清单 4-5 中的第 34 行代码，可以得到如图 4-10 所示的结果，在这里面可以找到模型参数的估计值和估计值对应的标准差。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.962
Model:                            OLS   Adj. R-squared:                  0.960
Method:                 Least Squares   F-statistic:                     460.5
Date:                Sat, 22 Apr 2017   Prob (F-statistic):           2.85e-14
Time:                        15:13:47   Log-Likelihood:                -31.374
No. Observations:                  20   AIC:                             66.75
Df Residuals:                      18   BIC:                             68.74
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.9495      0.934     -1.017      0.323      -2.912       1.013
x              1.0330      0.048     21.458      0.000       0.932       1.134
==============================================================================
Omnibus:                        0.745   Durbin-Watson:                   2.345
Prob(Omnibus):                  0.689   Jarque-Bera (JB):                0.673
Skew:                           0.074   Prob(JB):                        0.714
Kurtosis:                       2.113   Cond. No.                         66.3
==============================================================================
```

1 数值大于0.05，应该被舍弃

2 95%的可能，参数所在区间

参数估计值

估计值的标准差

图 4-10

（1）小郭首先分析图中的标记 1。结果显示，虽然参数b的估计值为-0.9495，但是这个值在$b = 0$这个假设下的 P-value 却高达 32.3%。换句话说，根据现有的数据推断：虽然估计值为-0.9495，但参数b的真实值等于 0 的概率为 32.3%[20]。在统计学上，对于这种情况，我们认为参数b是不显著的（在统计学上，P-value 小于 5%称为显著，小于 1%称为极显著），在建模时，应该舍弃此参数，不把它纳入模型[21]。同理，参数a的估计值为 1.033，而且相应的 P-value 在保留 3 位小数时等于 0。也就是说 P-value 小于 1%，参数a是极显著的，应该被纳入模型。

（2）综合第（1）步中的分析，模型应被修改为如下的形式：

$$y_i = ax_i + \varepsilon_i \tag{4-15}$$

（3）然后小郭再来分析图中的标记 2。这里有两个数字，它们构成的区间被称为参数的置信区间。这个置信区间对应的置信度为 95%，也就说在 95%的情况下，参数a的真实将在[0.932, 1.134]区间内（真实值 1 就落在这个区间内）。同样地，可以得到参数b的 95%置信区间为[−2.912, 1.013]。这个区间包含 0，而参数b并不显著。从数学上可以证明这两者是等价

[20] 这句话，数学上并不严谨。严格的表达应为：在参数b等于 0 的假设下，如果拒绝这个假设，犯错的概率为 32.3%。然而这种类似哲学拷问的表述让人难以理解，所以把它近似为正文中的表述，虽然损失了一点准确性，但也未尝不可。

[21] 在绝大多数应用场景里，即使截距项的系数是不显著的，也不会把线性回归模型里的截距项删去。这样处理能使模型在数据平移和数据单位转换时保持稳定，但具体的细节较复杂，在此就不再赘述。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

的。也就是说，我们可以通过参数的 95%置信区间是否包含 0 来判断其是否显著（如果包含，则不显著；如果不包含，则显著）。

（4）除了如图 4-10 的结果外，小郭还通过函数 f_test 来做假设检验。比如调用程序清单 4-5 中的第 36、38 行代码来检验参数a, b是否显著；又比如调用第 40 行代码来检验“$a = 1, b = 0$”这两个假设同时成立的显著性。具体结果如图 4-11 所示。

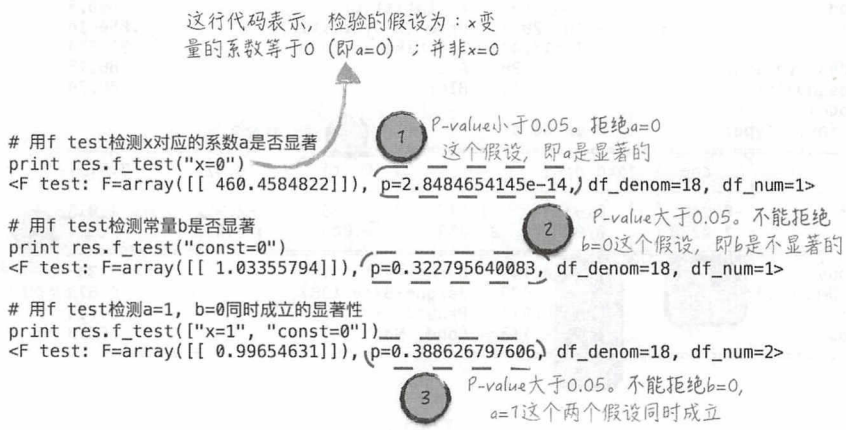


图 4-11

（5）读者可能注意到，图 4-11 中标记 2 的结果同图 4-10 中标记 1 的结果是相同的。但其实，图 4-10 展示的是 t_test 的结果。而图 4-11 则是 f_test 的结果。这两种检验的定义细节和差异，限于篇幅，这里就不做展开了。只想提醒读者其中最重要的一点：t_test 只能做单个变量的假设检验，比如“const=0”，但没法做多个变量的多个假设检验，比如“x=1”和“const=0”同时成立。而 f_test 两种都可以。

小郭根据上面的分析，按照公式（4-15）修改自己的模型并重新估计模型参数。我们来看看新模型的效果。

（1）正如前面提到的那样，线性回归模型的预测结果也是一个随机变量，同样可以计算模型预测结果的置信区间，如图 4-12 所示。虽然模型的预测值离真实值还有些距离，但所有的真实值都落在预测结果 95%的置信区间内。

（2）在实际生产中，通常有两种使用模型结果置信区间的方法：对未知数据做预测时，利用置信区间，确定真实值的大致范围。对数据做异常检测时，利用置信区间，筛选出那些在区间外的异常值。

（3）构建新模型的代码很简单，不加入常量变量即可，如程序清单 4-6 中的第 6 行代码所示。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

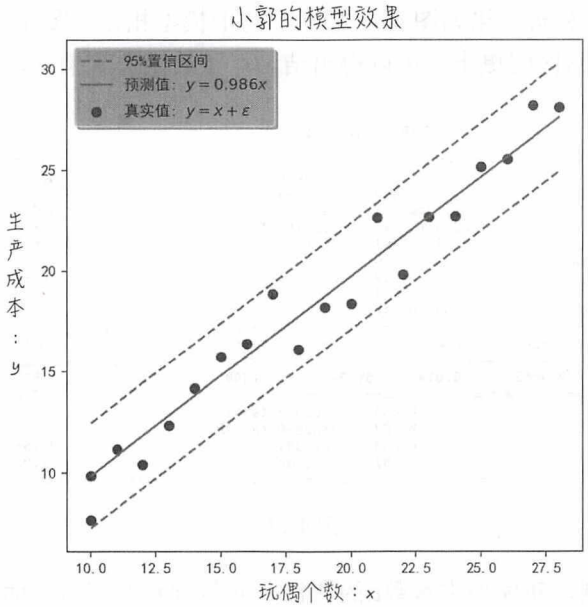


图 4-12

（4）计算预测结果的置信区间也很方便，使用 Statsmodels 提供的 wls_prediction_std 函数可以得到预测结果的标准差，置信区间上界以及置信区间下界，如第 17 行代码所示。其中的参数“alpha”将决定区间的置信度，即置信度为“1−alpha”。

（5）可视化这部分的其他代码，这里就不做展示和讨论了，请读者参考随书配套的代码。

程序清单 4-6 搭建新模型

```
1  from statsmodels.sandbox.regression.predstd import wls_prediction_std
2
3  def linearModel(data):
4      ### # 略去前面的代码
5      # const 并不显著，去掉这个常变量
6      resNew = trainModel(data[features], Y)
7      # 输出新模型的分析结果
8      print resNew.summary()
9      # 将模型结果可视化
10     visualizeModel(resNew, data, features, labels)
11
12 def visualizeModel(res, data, features, labels):
13     """
14     模型可视化
15     """
16     # 计算预测结果的标准差，预测下界，预测上界
17     prstd, preLow, preUp = wls_prediction_std(res, alpha=0.05)
18     … # 略去后面的代码
```

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

（6）新模型的统计分析结果如图 4-13 所示。与旧模型相比（图 4-10），新模型的的决定系数更大；参数a的置信区间更小。可以得出结论：新的模型效果更好。

新模型的统计结果

```
==============================================================================
Dep. Variable:                      y   R-squared:                       0.996
Model:                            OLS   Adj. R-squared:                  0.996
Method:                 Least Squares   F-statistic:                     4876.
Date:                Sun, 23 Apr 2017   Prob (F-statistic):           2.26e-24
Time:                        13:01:21   Log-Likelihood:                -31.933
No. Observations:                  20   AIC:                             65.87
Df Residuals:                      19   BIC:                             66.86
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x              0.9862      0.014     69.825      0.000       0.957       1.016
==============================================================================
Omnibus:                        0.489   Durbin-Watson:                   2.218
Prob(Omnibus):                  0.783   Jarque-Bera (JB):                0.561
Skew:                           0.033   Prob(JB):                        0.755
Kurtosis:                       2.182   Cond. No.                         1.00
==============================================================================
```

决定系数更大

置信区间更小，更准确

图 4-13

相信读者可以看到，新模型中参数a的置信区间包含真实值 1，而且它是更好、更精确的置信区间。新的模型为$y_i = 0.986x_i + \tilde{\varepsilon}_i$，这与真实的$y_i = x_i + \varepsilon_i$更加接近。也就是说，统计中的显著性分析帮助小郭搭建了更准确的模型。

把上面小郭的工作流程总结一下：他首先通过初步分析，假设一个最初的模型并编程实现它；再通过模型的分析结果，修正假设并得到最终的模型。通过小郭这个例子，我们可以发现，搭建模型是一个需要根据数据和模型结果，不断修正，迅速迭代的过程，正如第 2 章中图 2-2 一样。

4.3 模型陷阱

在数据科学的实践中，我们搭建模型通常有两个目的：使用模型对未知数据做预测；或者利用模型分析数据，找出数据的内在规律，为决策提供支持。在利用模型做预测时，我们希望模型的准确度越高越好，但在利用模型做分析时，我们则更关心模型是否是可靠的。在这两种实践中，分别会碰到如下的两类问题：

数据科学家在搭建模型时，通常会定义一些技术指标来衡量模型预测的准确度，比如在 4.1.1 节中，小陈针对线性回归，定义了均方差和决定系数。在利用历史数据训练模型时，这些技术指标都看起来不错，让我们对模型效果充满了信心。但当大家满心期待地把模型运用到生产上，真正对未知数据做预测时，却发现模型的效果并没有预想的那么好。有时候甚至很差，差到还不如随机乱猜。总结一下就是，模型的预测效果不稳定，实际准确度与预估

仅供非商业用途或交流学习使用

值相比，相差甚远。

除了做预测，分析已有数据也是模型很重要的功能。在大数据时代，有一个很重要的概念叫作数据驱动：传统上，公司的决策大多依赖于领导个人的经验判断。而在一个数据驱动的公司里，每个重大决定都将基于模型对数据的分析结果。举个简单的例子，运动鞋生产商 *A* 打算为自己的产品投放广告。而供它选择的有两种广告投放渠道：互联网和电视频道。互联网的主要受众是青少年，而电视的主要受众则是中年人。由于预算限制，公司需要根据产品的目标客户群，从中选择一个最优的广告渠道。为了准确地做出这个决定，*A* 公司的数据科学家利用历史数据对客户的购买意愿进行建模。他发现在模型中，年龄变量对应的参数是正数。这表示年龄越大，购买的几率也就越高。也就是说，它们产品的目标客户群是中年人，那么相应地应该选择电视这个渠道。

在这种情形下，就要求模型参数的估计值是可靠的。但事实并非总是如此，比如在 4.2.1 节中，小陈的模型结果表示：玩偶生产的固定成本为负数。这显然与事实相违背。总结一下就是，模型能没抓住数据真正的内在关系，错误地估计被预测值与自变量之间的联动效应，即对应的参数。

这两类问题是如何产生的呢？是否有相应的方法去防范这些问题的发生？当然方法是有的，但为了更好地理解这些方法，我们先来看看这两类问题产生的原因，如图 4-14 所示。

在搭建模型的过程中，为了提高预测的准确度，数据科学家们常常从已知的特征中提取更多新的特征，并以此搭建复杂的模型。特别是随着深度学习这一概念的传播，大家似乎都热衷于使用复杂度更高的模型，即使面对的是很简单的场景。但是模型越复杂，越容易使其本身掉入不断“自我催眠、强化偏见”的陷阱，从而引起过度拟合的问题。一旦发生了过度拟合，模型越复杂，其实错得也就越多。这种情况下，训练模型时，各项评估指标都很好，但是一用模型就傻眼了。

在大数据时代，我们能获得比以前更多的变量，这给了我们更多的建模选择。在建模实践中，数据科学家们会寻找新的可能与被预测值相关的自变量，并将它们加入模型中。但由于模型训练从本质上讲，只是一些数字的运算。即使将毫不相关的变量放到模型里，也会得到相应的参数估计，而且这个估计值几乎不可能等于 0。这就造成了所谓的模型幻觉[22]：表面上得到了很多变量间的联动效应，但事实上这些效应并不存在，只是由随机变量引起的数字巧合。模型幻觉会造成分析结果，特别是模型参数的不可靠。它不仅会把不存在的效应估计为存在，而且更严重的是，新加入的变量有可能将原本比较正确的估计扭曲为错误，比如将模型中原来变量的正效应估计为负效应（变量对应的参数为正时，称为正效应，否则为负效应）。

[22] 在已有模型中，加入新变量除了会引起这里提到的模型幻觉外，还可能引起其他更严重的问题，比如共线性、内生性等。这些问题的细节将在第 7 章中详细讨论。

仅供非商业用途或交流学习使用

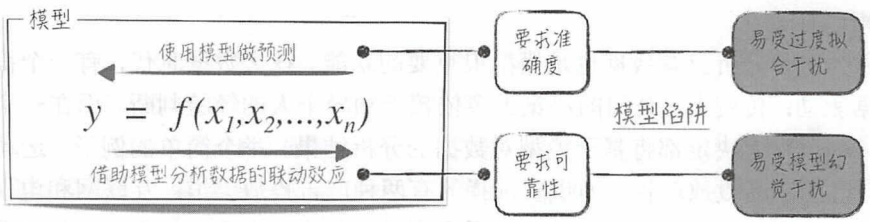


图 4-14

上面提到的过度拟合和模型幻觉并不是相互独立的。恰恰相反，它们常常交织在一起，相互加强，影响模型的准确度和可靠性。对于它们有对应的成熟解决方法，下面借助小陈和小郭的示例，分别来说明这些方法[23]。

4.3.1 过度拟合：模型越复杂越好吗

小潘提供的数据里，虽然只有一个自变量：玩偶个数$\boldsymbol{X} = \{x_i\}$。但小陈觉得只用一个变量的模型太简单了。他希望能从已知的变量中提取更多的特征供模型使用。于是，小陈使用了最常见的多项式变换来提取新的特征。具体来讲，小陈将原始数据$\boldsymbol{X}$转换成一系列特征$(\boldsymbol{X}, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^n)$，并使用它们搭建线性回归模型，如公式（4-16）所示，记为$\boldsymbol{Y} \sim (\boldsymbol{X}, \boldsymbol{X}^2, \ldots, \boldsymbol{X}^n)$。

$$y_i = b + a_1 x_i + a_2 x_i^2 + \ldots + a_n x_i^n \tag{4-16}$$

模型的结果如图 4-15 所示。从图上的技术指标来看，似乎特征越多、模型越复杂，效果就越好。当然读者知道，这个结论并不对，反而是图中最简单的模型最接近事实。为什么会出现这种情况呢？原因有以下两方面。

（1）从数学上来讲，向模型中加入一个新变量，那么原模型就成为了新模型的一个特例，即新变量的系数为 0。那么对于同一批已知数据，新模型的拟合效果一定比原模型好。但是模型对已知数据拟合得好并不代表它能对未知数据做出准确预测。

（2）从数据上来看，在现实中，我们使用已知的历史数据训练模型，但训练好的模型需要对现在未知的数据（未来的数据）做预测。也就是说，真正使用模型的数据在模型训练时，是不可见的。而这里，小陈使用全部现有数据训练模型，然后用模型对同样一批数据做预测并以此来评价模型效果。也就是说，评价模型效果的数据在训练模型时，对模型就是可见的。可以形象地理解为数据“既当裁判，又当运动员”，与现实严重不符。

[23] 示例的实现请参考随书配套的代码/ch04-linear/simple_example/linear_overfitting.py、linear_illusion_CI.py 和 linear_illusion_reg.py。

仅供非商业用途或交流学习使用

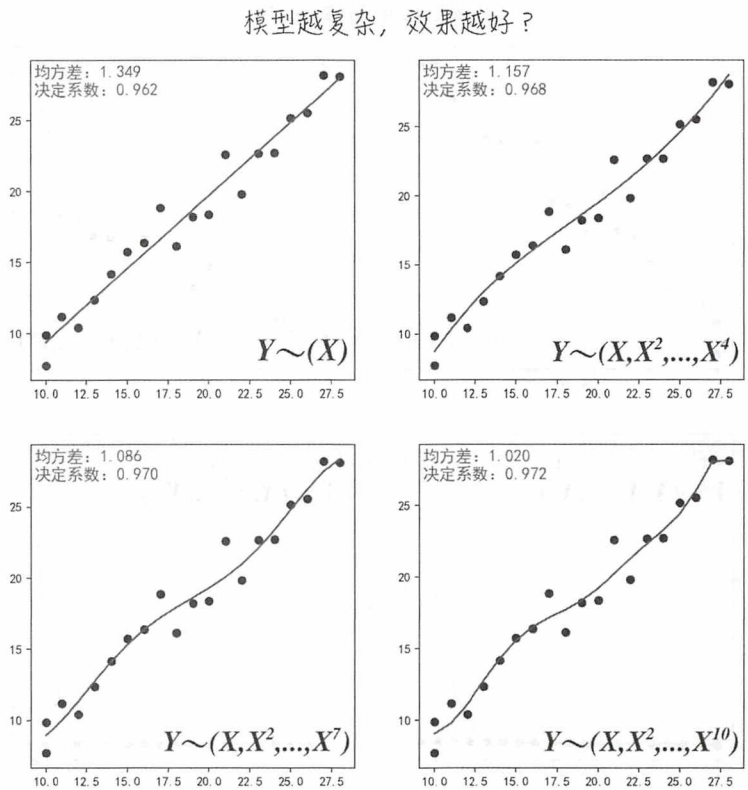


图 4-15

这两个原因使得小陈掉入了过度拟合的陷阱：搭建的模型被随机因素所干扰，而忽略了数据真实的内在规律，最终得到了错误的结论。

为了避免这个问题呢，可以将训练模型的数据和评价模型的数据分开。在训练模型时，评价模型的那部分数据是不可见的，这样才能正确地估计模型效果。正因为如此，在数据科学实践中，通常第一步操作就是将数据集分为两部分，分别叫作训练集（train set）和测试集（test set）。先利用训练数据估计模型参数，再使用测试数据估算模型效果，如程序清单 4-2 中第 12～17 行代码所示。这个方法在学术上被称为交叉验证（cross validation）。如果对上面的模型做相似的处理操作，得到的结果将如图 4-16 所示。结论与刚刚的完全相反，效果最好的是最简单的模型，这才是正确的结论。

将上面例子的结果上升到理论高度，方便我们在更一般的情况下处理过度拟合的问题。在学术上，将模型在训练集里的错误称为训练误差（training error），而模型在测试集里的偏差称为测试误差（testing error）。这两种误差与模型复杂程度的关系如图 4-17 所示，其中实现表示模型的训练误差，而虚线表示测试误差。

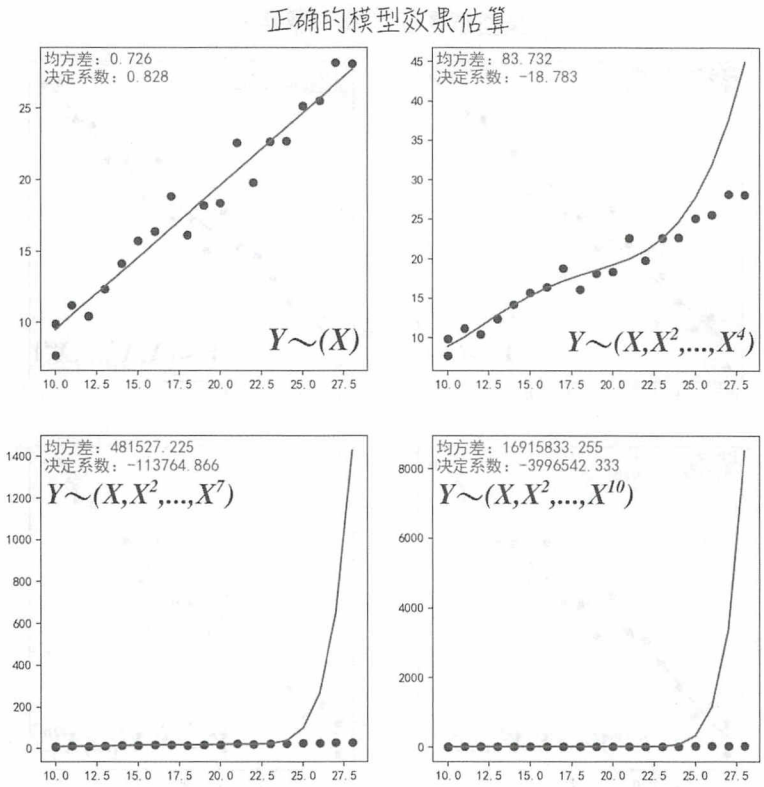

正确的模型效果估算
均方差：0.726
决定系数：0.828
Y～(X)
均方差：83.732
决定系数：-18.783
Y～(X,X²,...,X⁴)
均方差：481527.225
决定系数：-113764.866
Y～(X,X²,...,X⁷)
均方差：16915833.255
决定系数：-3996542.333
Y～(X,X²,...,X¹⁰)

图 4-16

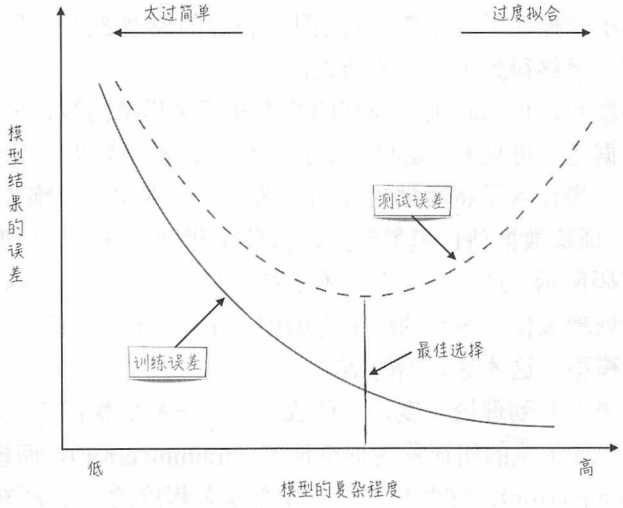

太过简单
过度拟合
模型结果的误差
测试误差
训练误差
最佳选择
低
高
模型的复杂程度

图 4-17

- 当模型太过简单时，无论是训练误差还是测试误差都很高，也就是说太过简单的模型还不足以捕捉数据里的复杂关系。
- 当模型太过复杂时，训练误差很小，但测试误差却相当高，这就是过度拟合。与模型太过简单相比，过度拟合是更加危险的，因为它具有极强的迷惑性，容易让人误以为模型的结果很理想。

在搭建模型时，上面的两种问题都会导致模型的效果不好，因此需要选择复杂程度合适的模型，使测试误差达到最小。

在实际的建模过程中，模型的复杂程度常常受模型幻觉的影响，因为它会引诱数据科学家在模型里加入新的特征，从而使模型变得更复杂。针对模型幻觉，统计学和机器学习都提供了各自的解决方案，下面就来讨论这些方案的细节。

4.3.2 模型幻觉之统计学方案：假设检验

对于小潘提供的数据，小郭在得到模型结果之后，觉得模型的自变量太少了，影响了模型的效果。所以，他着手寻找新的可能与小潘生产成本相关的变量。他找到了天气这个自变量，或许是否是晴天会影响手工制作的精度，从来影响小潘的生产总成本。于是，他引入了变量$\{z_i\}$，表示第i天是否是晴天，如果是，则$z_i = 1$，否则$z_i = 0$。加入这个变量后，小郭修改他的模型如下：

$$y_i = ax_i + bz_i + c + \varepsilon_i \tag{4-17}$$

但小郭不知道，天气这个变量是由另一个和玩偶制作完全独立的随机过程产生的。也就是说，天气对玩偶成本没有任何影响。从模型上来说就是：在公式（4-17）中，参数b的真实值等于 0。

小郭利用收集来的数据训练模型后，得到的结果如图 4-18 所示。

模型的决定系数下降

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.963
Model:                            OLS   Adj. R-squared:                  0.959
Method:                 Least Squares   F-statistic:                     222.8
Date:                Tue, 25 Apr 2017   Prob (F-statistic):           6.38e-13
Time:                        21:39:40   Log-Likelihood:                -31.141
No. Observations:                  20   AIC:                             68.28
Df Residuals:                      17   BIC:                             71.27
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
              coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.8983      0.953     -0.942      0.359      -2.910       1.113
x              1.0400      0.050     20.722      0.000       0.934       1.146
z             -0.3619      0.571     -0.634      0.535      -1.566       0.843
==============================================================================
Omnibus:                        1.295   Durbin-Watson:                   2.209
Prob(Omnibus):                  0.523   Jarque-Bera (JB):                0.852
Skew:                          -0.061   Prob(JB):                        0.653
Kurtosis:                       1.996   Cond. No.                         66.7
==============================================================================
```

1 b和c的估计值都不等于0

2 但b和c都不显著

3

图 4-18

虽然参数b, c的真实值都为 0，但模型给出的估计都不为 0，而且离 0 还都很远，这就是典型的模型幻觉。如果只根据参数的估计值，小郭会相信天气会影响玩偶的制作成本。他将建议小潘在晴天生产玩偶，这样玩偶的生产成本会降低−0.36，但这完全是一个荒谬的建议。

除此之外，正如上面提到的，由于无关变量的加入，模型幻觉还可能扭曲原有的，比较正确的估计。具体地，我们来看程序清单 4-7 这个示例。

程序清单 4-7 模型幻觉

```
 1 | import statsmodels.api as sm
 2 | import numpy as np
 3 |
 4 | def generateData():
 5 |     """
 6 |     生成模型数据
 7 |     """
 8 |     np.random.seed(5320)
 9 |     x = np.array(range(0, 20)) / 2
10 |     error = np.round(np.random.randn(20), 2)
11 |     y = 0.05 * x + error
12 |     # 新加入的无关变量 z 恒等于 1
13 |     z = np.zeros(20) + 1
14 |     return pd.DataFrame({"x": x, "z": z, "y": y})
15 |
16 |
17 | def wrongCoef():
18 |     """
19 |     由于新变量的加入，正效应变为负效应
20 |     """
21 |     features = ["x", "z"]
22 |     labels = ["y"]
23 |     data = generateData()
24 |     X = data[features]
25 |     Y = data[labels]
26 |     # 没有多余变量时，x 系数符号估计正确，为正
27 |     model = sm.OLS(Y, X["x"])
28 |     res = model.fit()
29 |     print "没有加入新变量时："
30 |     print res.summary()
31 |     # 加入多余变量时，x 系数符号估计错误，为负
32 |     model1 = sm.OLS(Y, X)
33 |     res1 = model1.fit()
34 |     print "加入新变量后："
35 |     print res1.summary()
```

如第 11 行代码所示，真实的模型是$y_i = 0.05x_i + \varepsilon_i$，其中$\varepsilon_i$服从标准的正态分布。而新加入的变量$z_i \equiv 1$，显然这个新加入的变量与被预测量$y_i$是相互独立的。我们分别只使用变量$X = \{x_i\}$建模，如第 27、28 行代码所示和同时使用$X = \{x_i\}, Z = \{z_i\}$建模，如第 32、33 行代码所示。得到的结果如图 4-19 所示。新模型将真实模型中的正效应错误地估计为负效应。

原模型：

	coef	std err	t	P>|t|	[0.025	0.975]
x	0.1001	0.046	2.197	0.041	0.005	0.195

根据新模型的结果，x对y是负效应，即y随着x的增加而减少。这是一个完全错误的结论

加入新变量之后：

	coef	std err	t	P>|t|	[0.025	0.975]
x	-0.0139	0.081	-0.172	0.866	-0.184	0.156
z	0.7221	0.432	1.671	0.112	-0.186	1.630

图 4-19

统计的假设检验或者与之等价的参数的置信区间，是解决模型幻觉问题的常用方法之一。具体的细节在之前 4.2.2 节里已经比较详细地讨论过了。使用同样的步骤，可以将多余的不相关变量排除出模型，比如小郭例子中的常数变量和天气变量。

4.3.3 模型幻觉之机器学习方案：惩罚项[24]

下面再次回到从事机器学习的小陈身上，看看他提供的模型幻觉的解决方案。回顾 4.1 节中，为了估计模型参数，小陈定义了损失函数L（小郭使用的最大似然法也类似）。那么对于小郭的新模型（公式（4-17）），对应的损失函数如公式（4-18）所示。

$$L = \sum_i (y_i - ax_i - bz_i - c)^2 \tag{4-18}$$

现在的问题是，对于真实值等于 0 的系数（比如b和c），它们的估计值不等于 0 或者不接近于 0。既然这是一个有数学公式引起的瑕疵，那么小陈希望能从数学上增加一种效应，使那些本该等于 0 的参数估计值尽量往 0 靠近。为了达到这个目的，小陈决定在原有的损失函数里加入惩罚项（或者叫作正则化项），将损失函数改写成如下形式：

$$L = \sum_i (y_i - ax_i - bz_i - c)^2 + \alpha(|a| + |b| + |c|) \tag{4-19}$$

其中新加入的损失$\alpha(|a| + |b| + |c|)$就是惩罚项[25]，其中，α表示惩罚的权重。可以看

[24] 这个方案在学术上叫 regularization（这个单词中文翻译为正则化），因此这个方案的标准名称为正则化。

[25] 在数学上，惩罚项$(|a| + |b| + |c|)$其实就是参数(a, b, c)的$L_1 - norm$，即 L1 范数。加入这个惩罚项后，对应的模型在学术上被称作 Lasso 回归。其实还有另一种很常用的惩罚项$(a^2 + b^2 + c^2)$，也就是参数(a, b, c)的$L_2 - norm$。$L_2 - norm$称为 L2 范数，是我们很熟悉的欧式距离。如果线性回归模型加入 L2 范数惩罚项，对应的模型在学术上被称为 ridge 回归。

对于这两种回归的差别，这里不做过多的讨论。只想说明，Lasso 回归更易于获得“稀疏”（sparse）解，即它求得的参数里会有更少的非零分量。

如程序清单 4-8 中的第 8 行代码，借助 fit_regularized 函数的参数“L1_wt”，我们可以综合这两种不同的回归：当“L1_wt=1.0”（默认情况）时，模型为 Lasso 回归；当“L1_wt=0.0”时，模型为 ridge 回归；当“L1_wt”在 0、1 之间时，为两种回归的加权平均。

到$\alpha > 0$时，惩罚项会随着a, b, c绝对值的增大而增大。通俗地讲，就是模型参数估计值越远离 0，惩罚就越大。因此在估计参数时，也就是寻找L的最小值时，这一项会迫使参数估计值向 0 靠近，而且跟$\{y_i\}$越不相关的变量，其对应的参数估计值向 0 靠近的步伐也越大。程序清单 4-8 展示了整个实现过程。

程序清单 4-8 加入惩罚项

```
1 | import statsmodels.api as sm
2 |
3 | def trainRegulizedModel(X, Y, alpha):
4 |     """
5 |     训练加入惩罚项的线性回归模型
6 |     """
7 |     model = sm.OLS(Y, X)
8 |     res = model.fit_regularized(alpha=alpha)
9 |     return res
```

第 8 行代码里面的“alpha”就是公式（4-19）中的α，即惩罚项的权重。模型参数的估计值将依赖于这个参数的取值。随着“alpha”值的增大，模型参数估计值的绝对值会逐步减少。整个过程如图 4-20 所示。当“alpha=0.1”时，模型参数的估计值为$a = 0.985, b = 0, c = 0$，这也是比较“正确的”参数估计值。

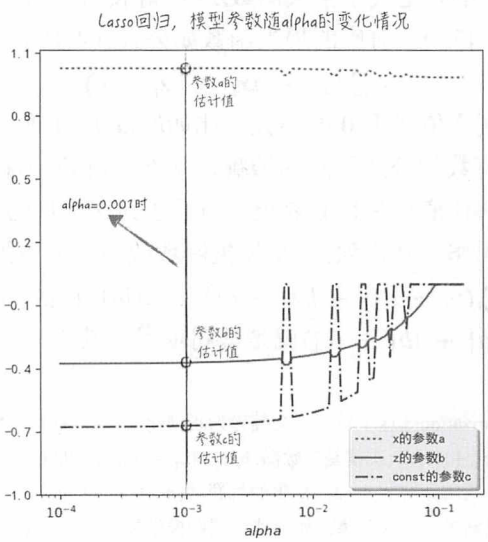


图 4-20

从上面的例子可以看到，α的取值影响模型参数a, b, c的估计，但是它又并不是模型本身的一部分。所以有必要将两者区分开来对待。在学术上，a, b, c被称为参数（parameter），而α被称为超参数（hyperparameter）。超参数通常被分为两类：一类是像α这种，用于修正模型

的；还有一类是工程实现上的超参数，这类超参数会在后面的章节里再做详细介绍。

超参数与参数不同，参数的估计值通常是有很明确的数学公式的。而超参数的估计就没那么严谨了，随意性较大，供数据科学家操作的空间也比较大，通常采用类似遍历的方法。具体来说就是事先给定一系列可能的超参数的值，这些给定的值依赖于数据科学家个人的偏好和经验。然后遍历给定的超参数集合，评估对应的模型效果，并从中选择效果最好的超参数。以上面的模型为例，这个过程如图 4-21 所示。在学术上，这个方法被称为网格搜寻（grid search）。

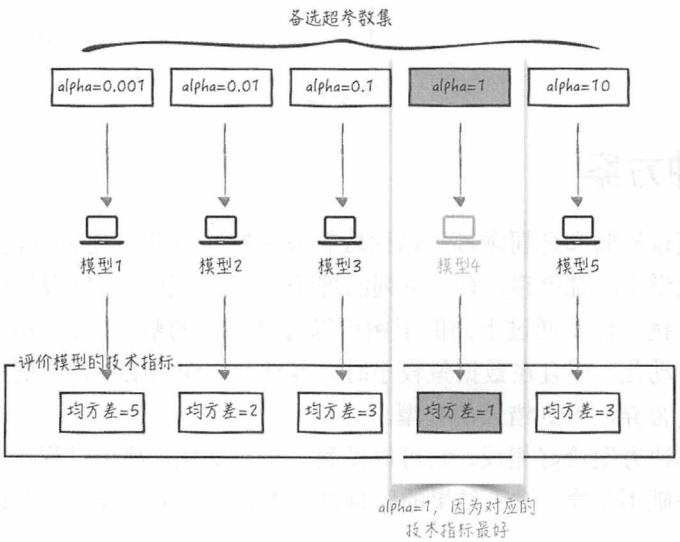


图 4-21

在 4.3.1 节中讨论过拟合问题时，我们介绍了一种防范过拟合的方法——交叉验证。这个方法将数据集分为训练集和测试集两部分，训练集专门用于估计模型参数，而测试集用于估算模型效果。但如果搭建的模型包含超参数，如上面的例子，那么这种做法就不再合适了。如果数据集依然被分为两部分，这表示我们将利用训练集估计参数，再根据模型在测试集里的效果选择超参数。也就是说原本应该用于评估模型的测试集，被误用于选择模型的超参数。这样一来，在训练模型时，原本应该被隔离的测试数据，也是部分可见的。这会导致模型效果被高估，进而引起过拟合的问题。

为了规避这个问题，需要保证测试数据在选择超参数时也是不可见的。于是，我们将数据分为 3 部分：训练集（train set）、验证集（validation set）和测试集（test set）。其中训练集用来估算模型参数，验证集用来选择模型超参数，而测试集用来评估模型效果。整个过程如图 4-22 所示。

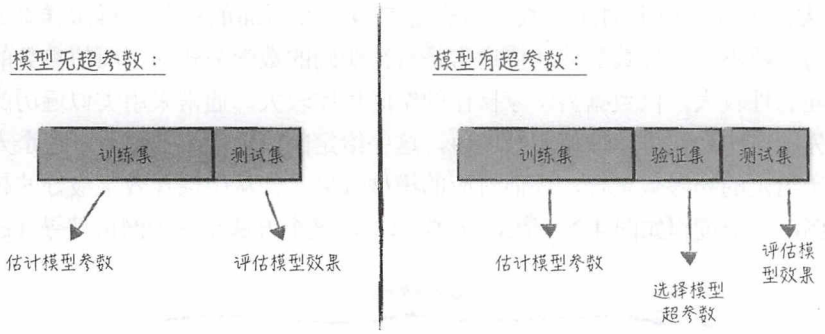


图 4-22

4.3.4 比较两种方案

上面提到的假设检验和惩罚项都可以比较有效地解决模型幻觉的问题。

假设检验在数学上更加严谨，有一系列的理论做支撑。我们可以根据检验结果完全排除掉不相关变量的干扰。但是通过上面的示例可以看到，它的整个过程需要较多的人为干预，并不能做到完全自动化。而且在数据量较小时，容易发生误判情况，即把相关变量判定为不相关的，进而将其舍弃，得到错误的模型。

惩罚项这个解决方案恰好相反，它可以做到完全自动化，整个过程不需要人为干预。但是缺点在于理论基础不牢靠，对于结果的解释性较差。很难用比较简单的语言将方案解释给非技术人员。

这两种方法各有优缺点，它们其实分别对应着第 1 章里讨论过的数据模型和算法模型。那在实际应用过程中，如果面对的场景是对数据建模，然后通过模型解释变量效应，这时最好选择假设检验这个方法。借助它能清楚地解释变量筛选的理由，而且通过整个分析过程，能更好地理解数据间的关联效应。如果面对的场景是为了做预测或者要求能将模型工程化，做到自动触发，自动学习，则需要使用加入惩罚项[26]这个方案。

4.4 模型持久化

之前的章节花了大量的篇幅来讨论如何搭建线性回归模型。虽然线性回归是非常简单的模型，但搭建好这个模型仍需要耗费大量的时间和精力。对于之后章节将要讨论的复杂模型，

[26] 在机器学习中，惩罚项是十分常用的技术。它不仅可以用来解决过拟合的问题，而且甚至成为了某些模型必不可少的部分，比如支持向量学习机（SVM），具体的细节请参考第 8 章。

搭建好它们更是一个工作量巨大的工程。不仅如此，在确定好模型框架后，训练模型（即估算模型的未知参数，具体的运算步骤和细节请参考第 6 章）也需要长时间的运算。特别是在大数据时代，在海量数据里训练模型甚至需要好几天的时间。

因此当模型训练好之后，我们希望能像普通程序一样，保存这些珍贵的劳动成果，以便之后有需求时，能再次复用之前训练好的模型。这个过程就是所谓的模型持久化，是这一节将讨论的主题。

4.4.1 模型的生命周期

在讨论如何保存模型之前，先来看看模型是如何度过“它的一生”。模型的“生命”可以分为明显不同的两个阶段——模型训练和模型使用。

- 在一个数据科学的项目中，我们通常先将收集到的历史数据存放在离线数据系统中，比如第 2 章里曾提到过的 Hadoop 集群。然后根据这些历史数据训练并挑选模型，并将选好的模型保存下来。
- 当新数据出现时，比如从网站上收集到的实时数据，使用之前保存好的模型对新数据做预测或者分析。而这些新数据本身将作为新的历史数据被存放到离线数据系统里。
- 经过一段时间的积累后，我们将再次触发模型的训练，利用更加全面、更有时效性的历史数据更新模型并保存，当然使用的模型也会随之更新。

由此可见，模型是一个不断循环、不断更新的过程，如图 4-23 所示。其中，模型的保存和读取是这个循环能否快速运转的关键。

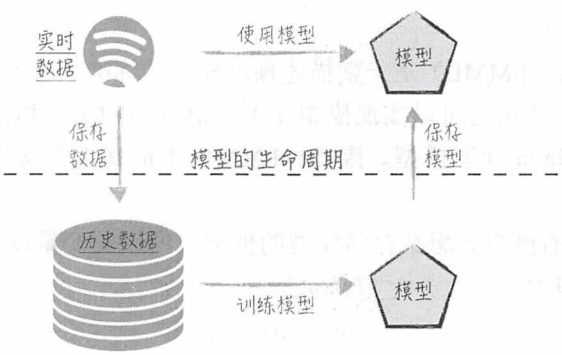


图 4-23

4.4.2 保存模型

由于我们使用 Python 搭建模型，那么相应的持久化方案有两种。如果模型的流转都在

Python 内部，也就是说，读取模型的程序也是 Python 程序，则可以使用内置库 pickle 来完成。如果模型的读取需要用到其他语言，比如生产中常用的 Java，则需要用到预测模型标记语言（Predictive Model Markup Language，PMML）。

pickle 是负责将 Python 对象序列化（serialization）和反序列化（de-serialization）的模块。换句话说，Python 中可以使用 pickle 将对象转化为文件保存在硬盘里，在需要的时候再读取并还原。如程序清单 4-9 中第 10 行代码所示，使用 pickle.dump 函数将训练好的模型保存到磁盘上（具体路径为“modelPath”）。当需要使用模型时，可以通过 pickle.load 函数读取保存在磁盘上的模型，如第 17 行代码所示。

程序清单 4-9 模型的保存与读取[27]

```
1  import pickle
2  from sklearn import linear_model
3
4  def trainAndSaveModel(data, modelPath):
5      """
6          使用 pickle 保存训练好的模型
7      """
8      model = linear_model.LinearRegression()
9      model.fit(data[["x"]], data[["y"]])
10     pickle.dump(model, open(modelPath, "wb"))
11     return model
12
13 def loadModel(modelPath):
14     """
15         使用 pickle 读取已有模型
16     """
17     model = pickle.load(open(modelPath, "rb"))
18     return model
```

预测模型标记语言（PMML）是一款描述预测模型的标准。这个标准与搭建和读取模型的具体编程语言无关。使用它可以实现模型在不同语言间的保存和读取，比如使用 Python 搭建模型，然后使用 Java 读取模型。因此，PMML 才是真正意义上实现了模型的存储和读取。

PMML 存储模型的核心思想是存储模型的框架、所用的变量以及模型里的参数。举个简单的例子，假设模型为：

$$y = ax + b \tag{4-20}$$

那么需要保存的信息为如下 3 项。

- 使用的模型为线性回归模型，如程序清单 4-10 中第 3 行所示。

[27] 完整的实现请参考随书配套的代码/ch04-linear/model_persistence.py。model_persistence.py。这个脚本使用了第三方库 sklearn2pmml，它的安装命令是“pip install --user git+https://github.com/jpmml/sklearn2pmml.git”。

- 模型的因变量为y，自变量只有一个x，如第 4～7 行所示。
- 模型参数a, b的取值，如第 8～10 行所示。

程序清单 4-10 PMML[28]

```
 1 | <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
 2 | <PMML xmlns="http://www.dmg.org/PMML-4_3" version="4.3">
 3 |     <RegressionModel functionName="regression">
 4 |         <MiningSchema>
 5 |             <MiningField name="y" usageType="target"/>
 6 |             <MiningField name="x"/>
 7 |         </MiningSchema>
 8 |         <RegressionTable intercept="-0.9495378313625586">
 9 |             <NumericPredictor name="x" coefficient="1.032966998995286"/>
10 |         </RegressionTable>
11 |     </RegressionModel>
12 | </PMML>
```

上面的例子只展示了 PMML 里的一小部分。在实际生产中，用 PMML 记录的信息比这个更加全面，比如我们会记录特征提取的方法等。

PMML 的实现比较复杂，好在有比较成熟的工具可以使用，其中最常用的是 JPMML（Java PMML API），读者可以在开源社区 GitHub 上找到这个项目。在实际生产中，它常见的使用方法如图 4-24 所示。具体的实现细节超出了本书的范围，在此就不做深入讨论了。

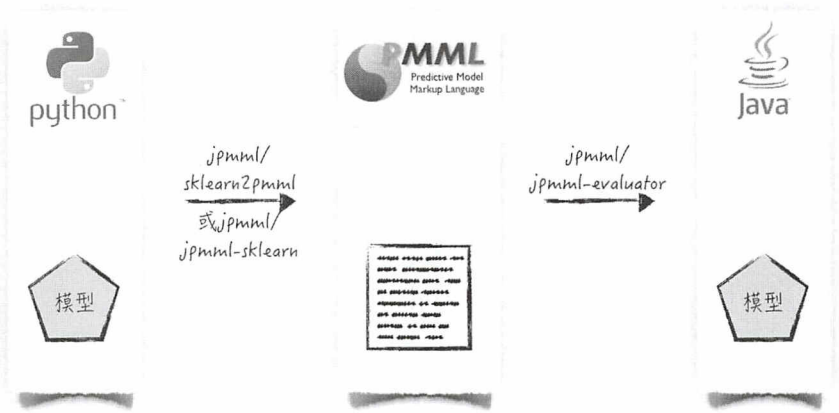


图 4-24

[28] PMML 利用 XML 描述和存储数据挖掘模型，是一个已经被 W3C 所接受的标准。程序清单 4-10 就是一个 PMML 的例子，只是由于篇幅限制，我们删去了其中不太重要的部分。

4.5 本章小结

本章开始介绍具体模型。我们通过一个简单的例子，主要讨论了线性回归模型；分别从机器学习和统计学的角度，着重介绍了模型假设、参数估计以及结果分析；并由此引出数据科学中很常见的两种错误：过度拟合和模型幻觉。对这两种错误，有相应的成熟解决方案：交叉验证、假设检验和惩罚项。这些方法不仅仅针对线性回归，对其他模型也是普遍适用的，是数据科学中非常重要的一部分。

线性回归模型的分析，特别是从统计学角度，需要很多很精细的数学推导。本书限于篇幅，这些数学细节往往被省略，而直接给出了结论。对细节有兴趣的读者可以参考其他书籍，比如 *Mathematical Statistics and Data Analysis*。

正如我们本章开篇所说的那样，线性回归是所有模型的基石。在接下来的章节里，我们将根据具体场景，逐个讨论如何基于线性模型搭建新的较为复杂的模型。

第5章

逻辑回归：隐藏因子

To be, or not to be, that is the question.
（生存还是毁灭，这是一个值得考虑的问题。）
——*William Shakespeare*

5.1 二元分类问题：是与否

5.2 上手实践：模型实现

5.3 评估模型效果：孰优孰劣

5.4 多元分类问题：超越是与否

5.5 非均衡数据集

5.6 本章小结

本章将讨论逻辑回归模型。这个模型从各个方面深刻地影响了我们的日常生活：小到上网时，网页播放的广告内容；大到国家各项经济政策的制定，比如扶贫政策。我们都可以发现逻辑回归的身影。

除了应用广泛，逻辑回归的建模过程体现了数据建模中很重要的思想：对问题划分层次，并利用非线性变换和线性模型的组合，将未知的复杂问题分解为已知的简单问题。毫不夸张地说，理解好逻辑回归的细节，就掌握了数据建模的精髓。

就其模型本身而言，有以下几个特点。

- 模型简单：建模思路清晰，容易理解与掌握。
- 适用范围广：模型的假设容易被满足，适用的场景很多。
- 模型可解释性强：模型参数有对应的实际意义，而且参数的取值直接反映特征的强弱，具有强解释性。
- 结果可靠性强：与线性回归模型类似，有很好的数学工具对模型参数和模型结果做分析，比如稳定性分析等。这使得模型结果的可信度很高。

下面就来具体讲解这些论断的原因。

5.1 二元分类问题：是与否

在现实中，我们常常面对一些二元选择，比如在逛街时，决定是否要购买眼前的这件衣服。不仅如此，很多事情的结果也是二元的，比如高考时，是否被某所大学录取。这些例子在生活中大量存在，也很有分析的价值。比如，某电商平台有一个预测模型，能很准确地预测客户是否会购买某件衣服。那么当客户打开这个电商的网页时，网站就可以根据模型的结果，把客户感兴趣的衣服通过网页广告推荐给他。这样既给客户提供了便利、提升了用户体验，同时也增加了电商平台的销售额。

既然在这些场景下，模型的用处这么大，那我们很想知道，如何针对这些场景进行建模：在第4章中介绍的线性回归模型是否适用于这些问题？很可惜，答案是否定的。因为从模型的角度来看，这是一个分类问题，比如将客户分为购买衣服和不购买衣服两类。它和第4章里讨论的回归问题有很大的不同，线性回归模型并不能很有效地解决这类问题。为了更直观地说明这一点，我们来看下面这个简单例子。

5.1.1 线性回归：为何失效

数据集里有两个变量：一个是自变量$\{x_i\}$，另一个是因变量$\{y_i\}$。其中因变量$\{y_i\}$的取值只有两个：0或1，记为$y_i \in \{0,1\}$。用学术一点的话来讲就是，y_i并不是一个连续值，而是一个离散值。数据这样设定是因为现在研究的场景是二元分类，即结果是二元的，因此在数

学上，就可以用 0/1 来表示这些结果，比如 1 表示购买衣服或被大学录取，而 0 表示不购买衣服或者没被大学录取。现在需要搭建模型，根据$\{x_i\}$对$\{y_i\}$做预测。因此对于场景类型，初步分析结果如图 5-1 所示。

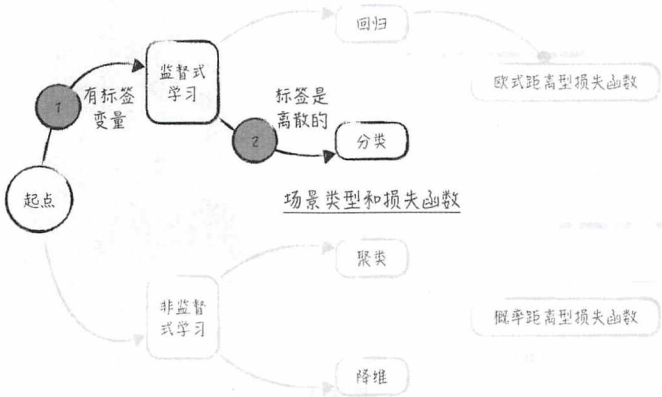


图 5-1

如果用线性回归对数据建模，得到的模型如下：

$$y_i = ax_i + b + \varepsilon_i \tag{5-1}$$

其中，ε_i是一个随机变量，它服从期望为 0，方差为σ^2的正态分布，记为$\varepsilon_i \sim N(0, \sigma^2)$。这是线性回归模型一个非常重要但又常常被人忽视的假设。

将数据画在坐标系中如图 5-2a 所示[1]。可以很直观地发现，数据并不能被近似地看成一条直线。如果用公式（5-1）表示的线性回归模型去拟合这些数据，得到的模型如图中的实线所示。模型预测值与真实值之间的误差较大，效果并不好。而且模型的预测结果显得有些莫名其妙：根据设定，y_i的值表示一种选择，所以只有 0/1 两个值。但模型预测值充满整个实数集，让人困惑。比如对某个y_i的预测为 0.73，这表示什么意思呢？

而且更重要的是，如果将模型误差（ε_i的观测值）的分布图画出来，会发现它并不符合正态分布。模型误差分布图如图 5-2b 所示。模型误差似乎是一个双峰分布，而正态分布是一个单峰分布。更严谨的数学推导可以证明，这里得到的模型误差的确不符合正态分布[2]。

---

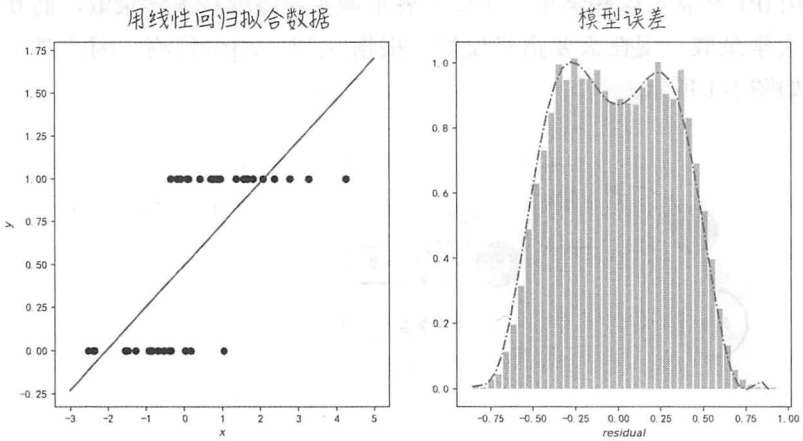
[1] 为了图形的展示效果更好，图 5-2a 中只展现了随机选取的部分数据点。剩下数据的分布情况于图中类似。图 5-2 和图 5-7 的具体实现请参考随书配套的代码/ch05-logit/logit_space_trans.py。

[2] 这里给出严谨的数学推导。为了书写方便，用矩阵的形式表示模型。假设模型中有 k 个自变量分别为$x_1, x_2, \ldots, x_k$；因变量为y；随机扰动项为ε。令$\boldsymbol{X_i} = (x_{1,i}, x_{2,i}, \ldots, x_{k,i}, 1), \boldsymbol{\beta} = (a_1, a_2, \ldots, a_k, a_{k+1})^{\mathrm{T}}$，其中，$i$表示第$i$组数据。模型可以表示为

$$y_i = \boldsymbol{X_i\beta} + \varepsilon_i$$

根据假设，$\boldsymbol{X_i}$与ε_i相互独立，则$P(\varepsilon_i \mid \boldsymbol{X_i}) = P(\varepsilon_i)$。也就是说，给定一个$\boldsymbol{X_i}$，$\varepsilon_i$也是服从正态分布的。

但根据模型公式：$\varepsilon_i = y_i - \boldsymbol{X_i\beta}$，即$\varepsilon_i = 0 - \boldsymbol{X_i\beta}$或者$\varepsilon_i = 1 - \boldsymbol{X_i\beta}$。这表示给定一个$\boldsymbol{X_i}$后，$\varepsilon_i$只可能取两个值。这显然不符合正态分布。



（a）　　（b）

图 5-2

这是一个很严重的问题，因为在使用线性回归模型时，我们实质上做了 3 个假设（其实还有更多假设，但这里只讨论其中最基础的 3 个）：

- 因变量$\{y_i\}$和自变量$\{x_i\}$之间是线性关系；
- 自变量$\{x_i\}$与干扰项ε_i相互独立；
- 没被线性模型捕捉到的随机因素ε_i服从正态分布。

但模型误差的分布情况却显示第 3 个假设显然不成立。也就是说，数据并不符合模型的假设，导致模型没办法自圆其说，因此预测效果不好。

这是搭建模型时非常重要的一点。因为在绝大多数情况下，从计算的角度来看，模型本身对数据并没有要求。任何数据放进模型里，都能得到相应的参数估计，进而可以使用得到的模型对数据做预测。但模型效果如何保证呢？保证模型效果的方法就是不断检验数据是否符合模型假设。如果符合，则模型的效果一定不差，否则就无从谈起了。用错误的假设强行建模，比如图 5-2 对应的例子，得到的一定是一个“错误而且无用”的模型。即使在某些情况下，比如训练集的数据量不够大，模型训练时，得到的评估指标还不错，那也只是一时的海市蜃楼。等到模型真正用武之时，就会发现模型的预测结果错得离谱。这也是为什么作者在前面的章节里一直强调，搭建模型需要不断提出假设和检验假设。

既然线性模型并不适用于这些场景，那应该如何搭建模型呢？

5.1.2 窗口效应：看不见的才是关键

以购买衣服为例，仔细分析整个场景会发现：能被其他人观察到的只有购买与否这个行

为。而这个行为其实是客户经过内心博弈之后的最终结果。也就是说，我们观察到的结果是经过一层变换后的结果，下面来详细解释。

先来看看购买衣服能给客户带来什么。一方面，购买衣服能给客户带来快乐，因为衣服能满足客户需求，比如爱美的需求或者保暖的需求等；但另一方面，购买衣服也会给客户带来烦恼，比如购买衣服需要花费金钱等。客户在选购衣服时，会综合这两方面因素，最终做出是否购买的决定。比如，当客户觉得衣服带给他的好处大于衣服的价格时，他就会购买。用比较专业的经济学术语来描述就是，购买行为能带给客户正效用，也能带给客户负效用。当正效用大于负效用时，也就是购买行为带来的整体效用大于 0 时，客户就会购买，反之则不然。

把上面这段文字翻译成模型的语言：假设有自变量集合$\boldsymbol{X_i} = (x_{1,i}, x_{2,i}, \ldots, x_{k,i}, 1)$，它们将决定购买衣服对客户$i$的效用，包括客户快乐的正效用$y_i^*$和使客户烦恼的负效用$y_i^{\sim}$。我们观察到的客户购买行为记为$y_i$，其中，$y_i = 1$表示客户$i$购买了衣服；$y_i = 0$，则表示客户$i$没有购买衣服。于是可以得到下面的公式：

$$y_i^* = f(\boldsymbol{X_i}), y_i^{\sim} = g(\boldsymbol{X_i})$$
$$y_i = \begin{cases} 1, & y_i^* > y_i^{\sim} \\ 0, & y_i^* \leqslant y_i^{\sim} \end{cases} \tag{5-2}$$

这时进一步假设，正效用和负效用都和自变量是线性关系。具体地，假设$\boldsymbol{\varphi} = (\varphi_1, \varphi_2, \ldots, \varphi_k, \varphi_{k+1})^{\mathrm{T}}$；$\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_k, \omega_{k+1})^{\mathrm{T}}$为模型参数，我们有：

$$y_i^* = \boldsymbol{X_i}\boldsymbol{\varphi} + \theta_i$$
$$y_i^{\sim} = \boldsymbol{X_i}\boldsymbol{\omega} + \tau_i \tag{5-3}$$

其中，θ_i, τ_i是相互独立的随机变量，且都服从正态分布。令$z_i = y_i^* - y_i^{\sim}$、$\boldsymbol{\gamma} = \boldsymbol{\varphi} - \boldsymbol{\omega}$以及$\varepsilon_i = \theta_i - \tau_i$（注意$\varepsilon_i$也服从正态分布，因为$\theta_i, \tau_i$相互独立），可以得到：

$$z_i = \boldsymbol{X_i}\boldsymbol{\gamma} + \varepsilon_i$$
$$y_i = \begin{cases} 1, & \boldsymbol{X_i}\boldsymbol{\gamma} + \varepsilon_i > 0 \\ 0, & else \end{cases} \tag{5-4}$$

进一步可以得到：

$$P(y_i = 1) = P(z_i > 0) = P(\varepsilon_i > -\boldsymbol{X_i}\boldsymbol{\gamma})$$
$$P(y_i = 1) = 1 - P(\varepsilon_i \leqslant -\boldsymbol{X_i}\boldsymbol{\gamma}) = 1 - F_\varepsilon(-\boldsymbol{X_i}\boldsymbol{\gamma}) \tag{5-5}$$

其中，F_ε是随机变量ε的累积分布函数，而$P(y_i = 1)$表示客户购买的比例。公式（5-5）对应的模型在学术上被称为 probit 回归（注意：虽然这个模型的名字里包含“回归”两个字，但它解决的是分类问题），整个过程可以近似为如图 5-3 所示的样子。

在模型搭建的过程中，我们假设了没办法观测的变量：y_i^*和$y_i^{\sim}$，因此这类模型被称作隐含变量模型（latent variable model）；而y_i^*和$y_i^{\sim}$被称为隐含变量（latent variable）。

probit 回归在数学上是比较完美的，但遗憾的是，由于正态分布的累积分布函数比较复

杂，并没有解析表达式[3]。这使得 probit 回归模型的参数估计比较困难，限制了它的应用。为了使模型使用起来更加方便，需要对其做一些近似，让它在数学上更加简洁。

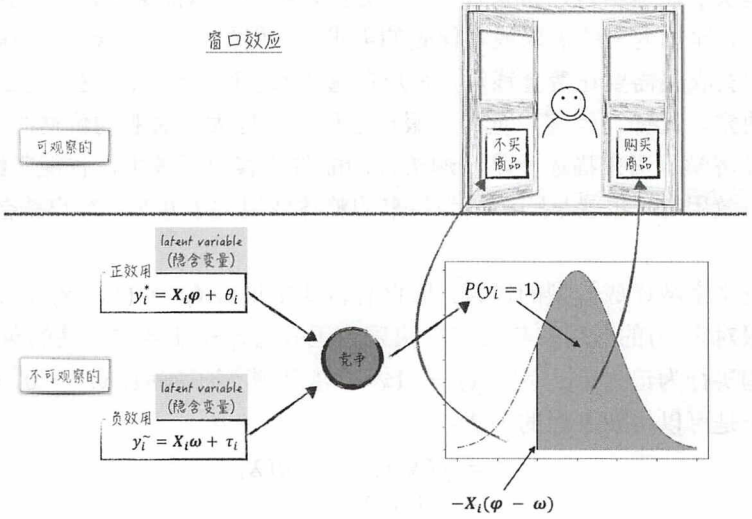


图 5-3

5.1.3 逻辑分布：胜者生存

由公式（5-5）可以看到，对 probit 回归做近似等价于对正态分布的累积分布函数$F_\varepsilon(x)$做近似。假设随机变量ε的期望为μ，方差为σ^2；又令函数$\phi(x)$为标准正态分布的累积分布函数。可以得到：

$$F_\varepsilon(x) = \phi\left(\frac{x-\mu}{\sigma}\right) \tag{5-6}$$

这说明正态分布在线性变换下保持稳定，因此，只需对标准正态分布做近似即可。数学家们发现逻辑分布可以很好地近似正态分布，具体效果如图 5-4 所示[4]。可以看到正态分布的累积分布函数几乎与逻辑分布的累积分布函数一样。

---

[3] 假设随机变量ε服从期望为μ，方差为σ^2的正态分布，则累积分布函数为：

$$F_\varepsilon(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{x} e^{-(t-\mu)^2/2\sigma^2}\,dt = \frac{1}{\sqrt{\pi}}\int_{-\infty}^{(x-\mu)/\sqrt{2}\sigma} e^{-t^2}dt$$

令$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}}\int_{-\infty}^{x} e^{-t^2}\,dt$，则$F_\varepsilon(x) = \frac{1}{2}\mathrm{erf}\left(\frac{(x-\mu)}{\sigma\sqrt{2}}\right)$。而$\mathrm{erf}(x)$函数是没有解析表达式的，所以正态分布的累积分布函数没有解析表达式。

[4] 图像的具体实现请参考随书配套的代码/ch05-logit/normal_logit_approx.py。

图中公式来源 Bowling S R, Khasawneh M T, Kaewkuekool S, et al. A logistic approximation to the cumulative normal distribution[J]. Journal of Industrial Engineering & Management, 2009, 2(1): 114-127.

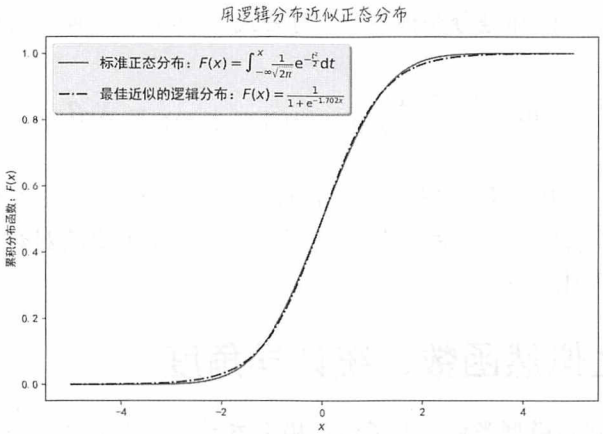


图 5-4

标准逻辑分布的概率密度函数为$f(x) = e^{-x}/(1 + e^{-x})^2$，对应的累积分布函数为：

$$S(x) = {}^{1}/_{(1 + e^{-x})} \tag{5-7}$$

公式（5-7）表示的函数在学术上被称为 sigmoid 函数，它在数据科学领域是一个非常重要的函数。特别是在神经网络和深度学习领域，我们会经常见到它。sigmoid 的函数图像如图 5-4 中的虚线，呈 S 形状，因此也常被称为 S 函数。从上面的分析可以得到：两种不同的效用（假定它们都满足线性回归模型的假设）相互竞争时，其中某一方最终胜出的概率分布在数学上可近似为 sigmoid 函数。通俗些讲，sigmoid 函数描述了某一方竞争胜出的概率。

将图 5-4 翻译成数学结果，可以得到：

$$F_{\varepsilon}(\sigma x + \mu) = \phi(x) \approx S(-1.702x) \tag{5-8}$$

因此，根据公式（5-5）和公式（5-8），略去一些繁琐的数学推导，可以得到公式（5-9）[5]：

$$P(y_i = 1) = {}^{1}/_{(1 + e^{-\boldsymbol{X}_i\boldsymbol{\beta}})}$$
$$P(\boldsymbol{Y} = 1) = {}^{1}/_{(1 + e^{-\boldsymbol{X}\boldsymbol{\beta}})} \tag{5-9}$$

公式（5-9）中的两个式子是等价的；第 1 个式子是对客户i购买概率的估计，而第 2 个公式是以矩阵的形式，将所有客户的购买概率写成一个表达式。其中，$\boldsymbol{X}_i = (x_{1,i}, x_{2,i}, \ldots, x_{k,i}, 1)$表示客户$i$的特征；$\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_k, \beta_{k+1})^{\mathrm{T}}$为模型参数。

公式（5-9）表示的就是逻辑回归模型（logit regression）。它与 probit 回归类似，虽然名

[5] 当然在公式（5-4）中，直接假设随机变量ε服从逻辑分布，也可以得到公式（5-9）。但是在作者看来，这样假设显得很随意，理论基础不牢，而且还漏掉了两种效用的竞争过程和对 sigmoid 函数的理解。这并不是一个很好的建模方式。

字里包含“回归”二字，但却是分类模型[6]。对公式（5-9）做变换，可以得到一个更有意思的公式：

$$\ln {P(y_i = 1)}/{[1 - P(y_i = 1)]} = \boldsymbol{X}_i\boldsymbol{\beta} \tag{5-10}$$

在统计上，$P/(1-P)$被称为发生比（odds），它表示该事件发生和不发生的比率。因此从公式（5-10）可以看到，逻辑回归其实是假设：事件发生比的对数为线性模型。这也是逻辑回归比较常用的简化版解释。

5.1.4 参数估计之似然函数：统计学角度

与第 4 章的线性回归模型类似，为了估计模型参数，需要定义一个合理的参数似然函数（或者模型损失函数）。然后根据这个函数，推导出模型参数估计值的表达式。

从统计学的角度来考虑这个问题，我们希望：根据模型，数据出现的概率越大越好（即最大似然估计法）。由于被预测值y_i是离散的，取值是 1 或者 0。所以y_i的概率分布函数可以写为

$$P(y_i) = P(y_i = 1)^{y_i}P(y_i = 0)^{1-y_i}$$

$$\ln P(y_i) = y_i \ln P(y_i = 1) + (1 - y_i)\ln[1 - P(y_i = 1)] \tag{5-11}$$

由于y_i之间相互独立，所以$P(\boldsymbol{Y}) = \prod_i P(y_i)$。再根据公式（5-9）中的第一个式子，可以推导出参数$\boldsymbol{\beta}$的似然函数L：

$$h(\boldsymbol{X}_i) = {1}/{(1 + \mathrm{e}^{-\boldsymbol{X}_i\boldsymbol{\beta}})}$$

$$L = P(\boldsymbol{Y} \mid \boldsymbol{\beta}) = \prod_i h(\boldsymbol{X}_i)^{y_i}[1 - h(\boldsymbol{X}_i)]^{1-y_i} \tag{5-12}$$

为了运算方便，通常对公式（5-12）中定义的似然函数取对数[7]，于是根据最大似然估计法可以得到参数估计值的公式：

$$\widehat{\boldsymbol{\beta}} = \mathrm{argmax}_{\boldsymbol{\beta}} L = \mathrm{argmax}_{\boldsymbol{\beta}} \ln L = \mathrm{argmax}_{\boldsymbol{\beta}} \ln P(\boldsymbol{Y} \mid \boldsymbol{\beta})$$

$$\widehat{\boldsymbol{\beta}} = \mathrm{argmax}_{\boldsymbol{\beta}} \sum_i y_i \ln h(\boldsymbol{X}_i) + (1 - y_i)\ln[1 - h(\boldsymbol{X}_i)] \tag{5-13}$$

5.1.5 参数估计之损失函数：机器学习角度

站在机器学习的角度，对于这种二元选择问题，我们希望模型预测的标签（是/否）与

---

[6] 这两个模型的命名如此奇怪是由于历史原因。生物学家 Chester Ittner Bliss 于 1934 年首次提出 probit 模型时，就取名为 probits regression；而作为 probit 模型改良版的 logit 模型，统计学家 David Cox 于 1958 年提出它时，沿用了 regression 这个名称，取名为 logit regression。

当然，这两个模型在很多处理细节上，比如参数的置信区间等，都沿用了线性回归的方法，所以称其为回归也未尝不可。只是读者需要注意，不要被它们的名字所迷惑就好。

[7] 对数函数是单调递增的，而且定义域为$(0, +\infty)$。

书籍是人类进步的阶梯

真实值越接近越好。借用上面购买衣服的例子，用数学的语言表达就是：假设模型预测将购买衣服的客户集合为A；真实购买衣服的客户集合为B，我们希望A和B的交集“越大越好”（是否真的越大越好？我们将在 5.3.1 节中再做详细讨论），如图 5-5 所示。

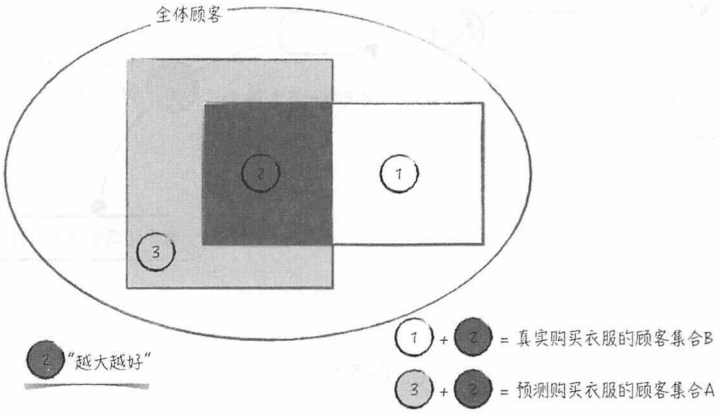


图 5-5

但根据这个描述，很难直接定义出一个在数学上方便处理的损失函数（不难看出，直接定义出来的损失函数不可导）。于是借鉴统计学中的做法，定义损失函数如下：

$$LL = -\ln L$$

$$LL = -\sum_i \{y_i \ln h(X_i) + (1 - y_i) \ln[1 - h(X_i)]\} \tag{5-14}$$

在学术上，公式（5-14）被称为交叉熵（cross entropy）。它可以被近似地看为两个概率分布之间的距离[8]，使估计的概率分布离真实的概率分布最近，等价于使损失函数LL达到最小值。所以不妨将LL称为概率距离型损失函数。至此，我们完成了场景分析和损失函数的定义，具体的结果如图 5-6 所示。参数估计值的具体公式与公式（5-13）完全等价，这里不再赘述。

[8] 在机器学领域，当需要度量两个概率之间的差异时，我们常常会用到所谓的 KL 散度（Kullback–Leibler divergence）。假设针对同一离散的随机变量x，有两个概率分布P, Q，P相对于Q的散度定义为：

$$D_{KL}(P \,||\, Q) = \sum_i P(i) \ln \frac{P(i)}{Q(i)}$$

如果x是连续的，定义类似。数学上可以证明$P = Q$时，两者之间的 KL 散度等于 0；而且它们差异越大时，对应的散度越大。另外，经过数学推导可以得到：

$$D_{KL}(P \,||\, Q) = H(P, Q) - H(P)$$

其中，$H(P, Q)$为两者的交叉熵；$H(P)$为一常数，称为P的熵。因此，交叉熵可以看成是两个分布之间的距离。

书籍是人类进步的阶梯

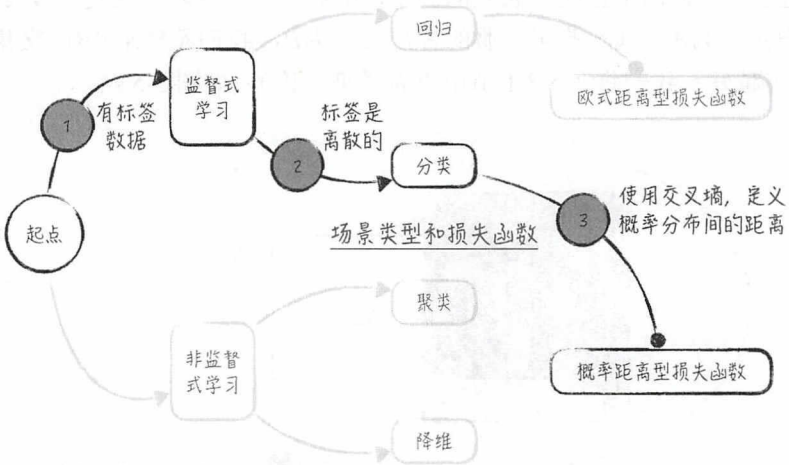


图 5-6

在实际应用中，为了防止过拟合，我们通常会在逻辑回归的损失函数（公式（5-14））中加入惩罚项。常见的惩罚项有 L1 范数，即模型参数的绝对值之和与 L2 范数，即模型参数的欧式距离。这与线性回归中的情况非常相似，具体的细节请参考 4.3.3 节。

5.1.6 参数估计之最终预测：从概率到选择

根据上面给出的公式（5-13），可以得到模型参数的估计值$\widehat{\boldsymbol{\beta}}$。将这个估计值代入公式（5-9），可以对客户$i$做出预测，得到他购买衣服概率的预测值，即

$$\hat{P}(y_i = 1) = {}^{1}/_{(1 + e^{-X_i\widehat{\boldsymbol{\beta}}})} \tag{5-15}$$

但这只是一个有关概率的预测值，而我们真正想要的是：对客户最终购买行为的预测。这两者之间还差最后一步。这是一个二元选择问题，在一般情况下，当$\hat{P}(y_i = 1) > \hat{P}(y_i = 0)$时（即$\hat{P}(y_i = 1) > 0.5$），预测客户会购买衣服是合情合理的。那么最终的模型结果可以写成：

$$\hat{y}_i = \begin{cases} 1, & 1/(1 + e^{-X_i\widehat{\boldsymbol{\beta}}}) > 0.5 \\ 0, & else \end{cases} \tag{5-16}$$

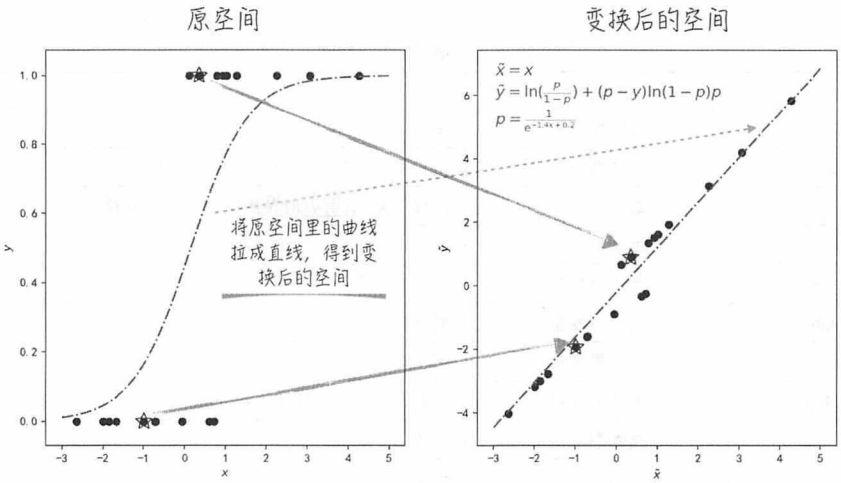
当然在一些特殊情况下，需要人为选定一个阈值α，当$\hat{P}(y_i = 1) > \alpha$时，才预测客户会购买衣服。这部分的细节讨论将在 5.3 节中展开。

5.1.7 空间变换：非线性到线性

在讨论具体的代码实现之前，我们先从线性空间的角度来看看逻辑回归模型做了什么。

书籍是人类进步的阶梯

回到 5.1.1 节的例子中。从直观的图像上来讲，二元选择问题不能用线性回归模型解决的原因是：数据并没有大致分布在一条直线周围，而是呈两条彼此分开的直线状。如图 5-7a 里的黑色圆点，如果用一条直线去近似这些圆点，效果不会太好。但如果形象地把图 5-7a（原空间）想象成橡皮泥，握住里面曲线的两头。用力将其拉成直线，得到图 5-7b（新空间）。在新的空间里，代表数据的黑点就几乎在一条直线上了。换句话说，在新的线性空间中，线性回归模型就可以很好地拟合数据了。



（a）　　（b）

图 5-7[9]

其实图 5-7a 中的曲线表示的就是 sigmoid 函数。因此，之前讨论的逻辑回归（公式（5-10）），对应在二维空间上，就是将图 5-7a 变换为图 5-7b，以便让数据落在一条直线周围。在高维空间也类似，只不过要将这里的直线换为超平面[10]。

这体现了机器学习算法中很重要的一点：通过非线性的空间变换（比如这里的 sigmoid 函数），将非线性问题转化为线性问题，再用线性模型去解决。在后面的章节，我们还会不断重复地看到这一过程，这也是作者会在第 4 章里说“线性回归是模型之母”的原因。

---

[9] 空间变换的公式来自于公式（5-14）中定义的交叉熵，所以图 5-7 左右两边的模型在数学上是等价的。

[10] 超平面（hyperplane）是一个数学概念，表示 n 维欧氏空间中，一个超平面用数学公式表示如下：

$$a_1x_1 + a_2x_2 + \ldots + a_nx_n = b$$

从模型的角度来讲，就是一个多元线性回归模型。在空间上来讲，是余维度等于一的线性子空间。这是平面中的直线、空间中的平面之推广。

书籍是人类进步的阶梯　　仅供非商业用途或交流学习使用

5.2 上手实践：模型实现

本节我们将结合具体的数据，讨论如何利用 Python，针对二元分类问题搭建逻辑回归模型[11]。所用的数据来源于美国加州大学欧文分校。它是美国个人收入的普查数据，具体的变量以及简单说明如表 5-1 所示。

表 5-1

变量名	变量类型	说明
age	数值型变量	年龄
workclass	类别型变量	工作类型，如公务员、私企职工等
fnlwgt	数值型变量	抽样权重。（普查时使用的变量，与建模分析无关）
education	类别型变量	学历，如本科、研究生等
education_num	数值型变量	受教育年限
martial-status	类别型变量	婚姻状况
occupation	类别型变量	所在行业
relationship	类别型变量	家庭角色，比如丈夫、妻子等
race	类别型变量	种族
sex	类别型变量	性别
capital_gain	数值型变量	该年度投资收益
capital_loss	数值型变量	该年度投资损失
hours_per_week	数值型变量	每星期工作时间
native_country	类别型变量	出生国家
label	类别型变量	年收入分类，分为两类：“>50K”和“<=50K”

其中年收入分类（label）是一个二元变量（只有两个可能的取值），也是需要被预测的变量。

5.2.1 初步分析数据：直观印象

数据集的自变量里有两类变量：一类是数值型变量，表示某项数值，比如年龄；另一类是类别型变量，表示所属类型，比如学历。这两者的区别在于如下两点。

- 数值型变量可以直接运算，比如 53 岁加上 2 岁等于 55 岁，相应的数学运算是有实

[11] 完整的实现请参考随书配套的代码/ch05-logit/example/logit_example.py；而所用数据为/ch05-logit/ example/data/adult.data。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

际意义的。

- 而类别型变量不能直接运算，比如高中学历和本科学历没法在数学上直接相加。即使通过某种变换，把类别型变量变成数字，对应的数学运算也是没意义的。例如用 1 表示高中学历，2 表示本科学历，3 表示研究生学历。根据数学，我们有$1+2=3$，但是实际上高中学历加本科学历并不等于研究生学历。

因此这两类变量有本质上的区别，在搭建模型是需要分别处理。这里为了专注于逻辑回归模型本身，先只使用数值型自变量，即年龄、受教育年限、该年度投资收益、该年度投资损失和每星期工作时间，整个选择过程如图 5-8 所示。

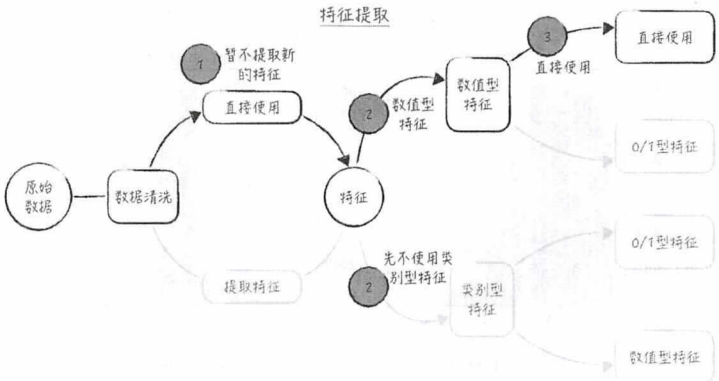


图 5-8

利用 Python shell 和 pandas 读取数据并选择想要的变量，见程序清单 5-1 所示。我们注意到被预测的变量“label”是文字型类别变量。因为文字并不能做数学运算，所以需要将其转换为数字型变量。

程序清单 5-1 读取数据——pandas

```
>>> import pandas as pd
>>> data = pd.read_csv(path)  # path 为数据存放的地址
>>> cols = ['age', 'education_num', 'capital_gain', 'capital_loss',
            'hours_per_week', 'label']
>>> data = data[cols]
>>> data.head(8)
   age  education_num  capital_gain  capital_loss  hours_per_week   label
0   39             13          2174             0              40   <=50K
1   50             13             0             0              13   <=50K
2   38              9             0             0              40   <=50K
3   53              7             0             0              40   <=50K
4   28             13             0             0              40   <=50K
5   37             14             0             0              40   <=50K
6   49              5             0             0              16   <=50K
7   52              9             0             0              45   >50K
```

书籍是人类进步的阶梯　　仅供非商业用途或交流学习使用

利用 pandas 可以很方便地将文字型类别变量转换为数字变量，并将变量的分布情况用图像展示出来。

（1）如程序清单 5-2 中第 1 行代码所示，在原数据上新生成一个变量“label_code”。这个变量有两个取值：0 表示“<=50K”、1 表示“>50K”。

（2）为了更加直观地了解变量的分布情况，可以使用直方图（histogram）将变量的分布情况可视化。如第 13 行代码，“data”是 pandas 里的 DataFrame 对象。利用它的 hist 函数，描绘年龄等变量的直方图。具体结果如图 5-9 所示，图像的横轴表示变量的取值区间，而纵轴表示样本落在对应区间的次数。

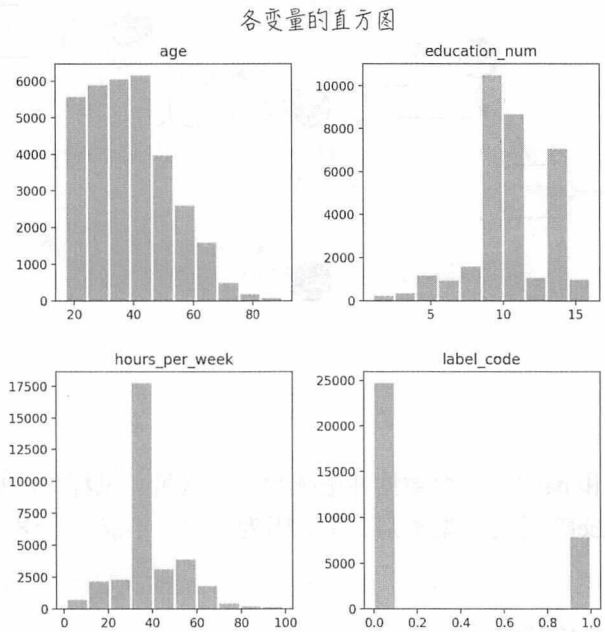


图 5-9

程序清单 5-2 数据转换及可视化——pandas

```
1  | >>> data["label_code"] = pd.Categorical(data.label).codes
2  | >>> data[["label", "label_code"]].head(8)
3  |      label  label_code
4  | 0   <=50K          0
5  | 1   <=50K          0
6  | 2   <=50K          0
7  | 3   <=50K          0
8  | 4   <=50K          0
9  | 5   <=50K          0
10 | 6   <=50K          0
11 | 7    >50K          1
```

书籍是人类进步的阶梯　　仅供非商业用途或交流学习使用

```
12 | >>> import matplotlib.pyplot as plt
13 | >>> data[["age", "hours_per_week", "education_num", "label_code"]].hist()
14 | >>> plt.show(block=False)
```

下面利用统计指标分析数据。

（1）如程序清单 5-3 中第 1 行代码所示，利用 DataFrame 的 describe 函数，可以得到数据的基本统计信息，如平均值、标准差等。值得注意的是，默认的 describe 方法只会显示数值型变量的统计信息。如果想要全部变量的统计信息，可以使用如下命令："describe (include='all')"。

程序清单 5-3 数据的基本统计信息——pandas

```
 1 | # 数据的基本统计信息
 2 | >>> data.describe()
 3 |          age  education_num  capital_gain  capital_loss  hours_per_week
 4 | count 32561.0        32561.0       32561.0       32561.0         32561.0
 5 | mean     38.6           10.1        1077.6          87.3            40.4
 6 | std      13.6            2.6        7385.3         403.0            12.3
 7 | min      17.0            1.0           0.0           0.0             1.0
 8 | 25%      28.0            9.0           0.0           0.0            40.0
 9 | 50%      37.0           10.0           0.0           0.0            40.0
10 | 75%      48.0           12.0           0.0           0.0            45.0
11 | max      90.0           16.0       99999.0        4356.0            99.0
```

（2）交叉报表（crosstab）是一种常见的数据分析方法。它可以用来描述两个变量之间的关系。如第 13 行代码所示，利用 crosstab 函数计算"education_num"和"label"的交叉报表。其中的"pd.qcut(data["education_num"], [0, .25, .5, .75, 1])"表示将变量"education_num"按分位数划分成 4 个区间。

（3）交叉报表的结果如第 15～20 行代码所示，报表的列表示"label"变量；而行表示"education_num"。报表对应位置的数字表示每种变量组合在样本中出现的频次，比如样本中有 1919 个数据点，它们的"label"为">50K"，同时，"education_num"属于[1, 9]区间。

（4）可以将上面得到交叉报表转换为图形，如第 23 行代码所示。得到的结果如图 5-10 所示，其中方块面积的大小与对应的表中数值成正比。从图中可以很容易地得到：随着受教育年限的增加，年收入大于 5 万的比例也随之增大。

程序清单 5-3 数据的基本统计信息——pandas

```
12 | # 计算 education_num, label 交叉报表
13 | >>> cross1 = pd.crosstab(pd.qcut(data["education_num"], [0, .25, .5, .75,
1]), data["label"])
14 | >>> print cross1
15 | label          <=50K   >50K
16 | education_num
17 | [1, 9]         12835   1919
18 | (9, 10]         5904   1387
19 | (10, 12]        1823    626
20 | (12, 16]        4158   3909
```

书籍是人类进步的阶梯　　仅供非商业用途或交流学习使用

```
21 | # 将交叉报表图形化
22 | >>> from statsmodels.graphics.mosaicplot import mosaic
23 | >>> mosaic(cross1.stack())
```

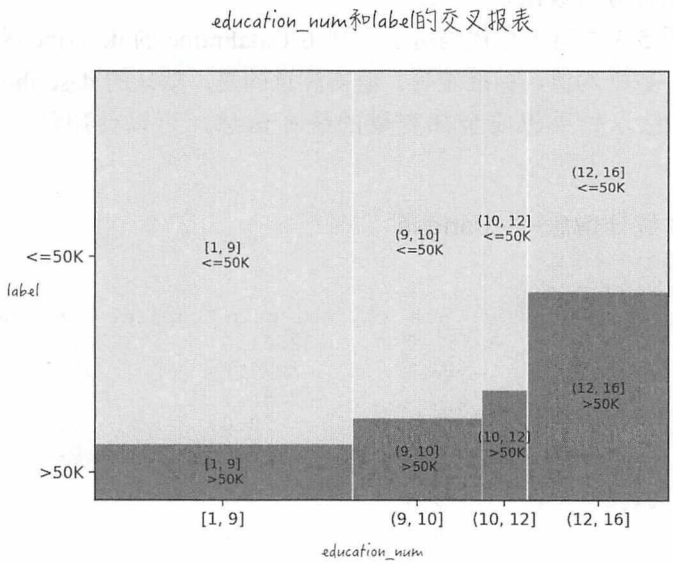


图 5-10

（5）类似地，可以计算“hours_per_week”与“label”的交叉报表。如第 25 行代码所示，其中“pd.cut(data["hours_per_week"], 5)”将变量“hours_per_week”按大小平均划分为 5 个区间。

（6）为了更好地了解每星期工作时间对年收入的影响，需要将上面得到的交叉表归一化。也就是说，交叉表中的数值等于该元素在对应行中的占比。如第 27 行代码所示，其中“cross2.sum(1).astype(float)”表示将“cross2”按行求和。

（7）将归一化后的数据图形化，得到如图 5-11 所示的图像。图中的结果告诉我们，年收入大于 5 万的比例并不总是与每星期工作时间成正比：当工作时间小于 80 小时，年收入大于 5 万的比例与工作时间成正比；而大于 80 小时之后，年收入大于 5 万的比例反而下降了。

程序清单 5-3 数据的基本统计信息——pandas

```
24 | # 计算 hours_per_week, label 交叉报表
25 | >>> cross2 = pd.crosstab(pd.cut(data["hours_per_week"], 5), data["label"])
26 | # 将交叉报表归一化，利于分析数据
27 | >>> cross2_norm = cross2.div(cross2.sum(1).astype(float), axis=0)
28 | >>> cross2_norm.plot(kind="bar")
29 | >>> plt.show(block=False)
```

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

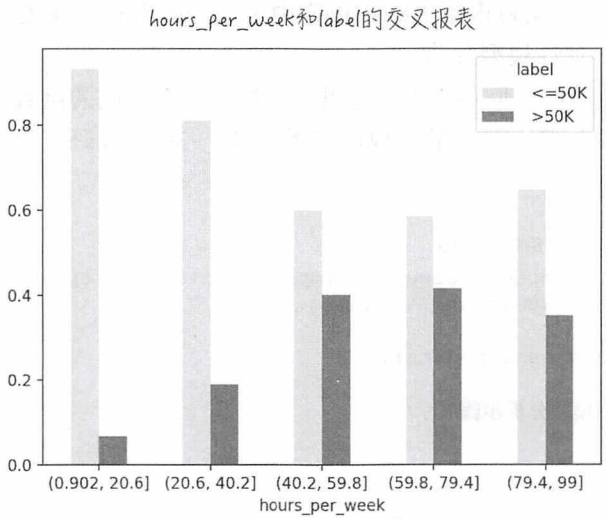


图 5-11

5.2.2 搭建模型

有了上面的准备工作和分析结果，我们就可以开始搭建模型了。

1．使用第三方库 Statsmodels 搭建模型

（1）如第 4 章中讨论的那样，为了避免过拟合的问题，需要将数据分为互不相交的两部分：一部分用来训练模型，剩下的则用来预估模型效果。如程序清单 5-4 中第 14 行代码所示，使用 scikit-learn 中提供的 train_test_split 函数将数据分为训练集和测试集，其中测试集占比 20%。

（2）使用 Statsmodels 搭建模型时，可以使用类似文字的表达式来构造模型[12]。如第 23 行代码所示，“formula”字符串就定义了模型的形式。下面先简单介绍一下模型定义的语法。

（3）为了表述方便，我们不妨假设一个更简单的模型表达式：“label_code ~ age + age * education_num ”。这个字符串对应的模型为 $label_code \sim a \times age + b \times age \times education_num + c$。具体来讲，字符“~”相当于等号，它左边的变量，即“label_code”，是被预测的变量。而它的右边是模型所用的特征，比如原生变量直接生成的特征“age”，又比如在原生变量的基础上，通过数学运算得到的新特征“age * education_num”。

（4）有了模型的定义字符串后，搭建并训练相应的模型就很简单了，如第 24、25 行代

[12] 如果读者对另一种常用的统计编程语言：R 语言比较熟悉，会发现 Statsmodels 中定义模型的语法和 R 语言中的几乎一摸一样。

仅供非商业用途或交流学习使用

码所示。其中，第 24 行代码将根据“formula”中定义好的形式创建逻辑回归模型；而第 25 行代码是在训练模型，估计模型参数。

（5）类似第 4 章中的线性回归模型，逻辑回归模型也有相同的函数接口来计算模型最常用的统计性质和做假设检验。具体的实现可参考第 33～39 行代码，返回的结果将在随后详细讨论。

程序清单 5-4 搭建模型——Statsmodels

```
1  from sklearn.model_selection import train_test_split
2  import statsmodels.api as sm
3
4  def logitRegression(data):
5      """
6      逻辑回归模型分析步骤展示
7
8      参数
9      ----
10     data : DataFrame，建模数据
11     """
12     ### # 略去前面的代码
13     # 将数据分为训练集和测试集
14     trainSet, testSet = train_test_split(data, test_size=0.2)
15     # 训练模型并分析模型效果
16     re = trainModel(trainSet)
17     modelSummary(re)
18
19 def trainModel(data):
20     """
21     搭建逻辑回归模型，并训练模型
22     """
23     formula = "label_code ~ age + education_num + capital_gain + capital_loss +
       hours_per_week"
24     model = sm.Logit.from_formula(formula, data=data)
25     re = model.fit()
26     return re
27
28 def modelSummary(re):
29     """
30     分析逻辑回归模型的统计性质
31     """
32     # 整体统计分析结果
33     print re.summary()
34     # 用 f test 检验 education_num 的系数是否显著
35     print "检验假设 education_num 的系数等于 0："
36     print re.f_test("education_num=0")
37     # 用 f test 检验两个假设是否同时成立
38     print "检验假设 education_num 的系数等于 0.32 和 hours_per_week 的系数等于
       0.04 同时成立："
39     print re.f_test("education_num=0.32, hours_per_week=0.04")
```

仅供非商业用途或交流学习使用

2. 利用统计学的方法分析模型参数估计值的稳定性

（1）调用程序清单 5-4 中的第 33 行代码，可以得到如图 5-12 所示的结果。与线性回归模型类似，返回的结果包含参数的估计值和对应的置信区间。图中的结果显示，模型中的每个参数都是显著的，因而没有变量需要被排除出模型。

（2）值得注意的是，变量“hours_per_week”对应的系数为 0.0399，大于 0。它表示：年收入大于 5 万的概率将始终随着每星期工作时间的增加而增大（这个结论的具体细节请见 5.2.3 节）。这个结论与图 5-11 中的图像不太一致。图像暗示，年收入大于 5 万的比例并不总是与每星期工作时间成正比。

```
                           Logit Regression Results
==============================================================================
Dep. Variable:             label_code   No. Observations:                26048
Model:                          Logit   Df Residuals:                    26042
Method:                           MLE   Df Model:                            5
Date:                Sun, 07 May 2017   Pseudo R-squ.:                  0.2639
Time:                        20:01:50   Log-Likelihood:                -10578.
converged:                       True   LL-Null:                       -14370.
                                        LLR p-value:                     0.000
==================================================================================
                    coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept        -8.2970      0.128    -64.623      0.000      -8.549      -8.045
age               0.0435      0.001     31.726      0.000       0.041       0.046
education_num     0.3215      0.008     42.231      0.000       0.307       0.336
capital_gain      0.0003   1.07e-05     29.650      0.000       0.000       0.000
capital_loss      0.0007   3.64e-05     20.055      0.000       0.001       0.001
hours_per_week    0.0399      0.001     26.995      0.000       0.037       0.043
==================================================================================
```

1 hours_per_week 的系数大于0

参数估计值

参数都显著

参数估计值的置信区间

图 5-12

除此之外，还可以通过 f_test 函数来做假设检验，比如调用程序清单 5-4 中的第 36 行代码来检验变量“education_num”的系数是否显著。具体的结果如图 5-13 所示。

这行代码表示，检验的假设为：变量education_num的系数等于0；并非education_num=0

```
检验假设education_num的系数等于0:
print re.f_test("education_num=0")
<F test: F=array([[ 1783.4276255]]), p=0.0, df_denom=26042, df_num=1>

检验假设education_num的系数等于0.32和hours_per_week的系数等于0.04同时成立:
print "检验假设education_num的系数等于0.32和hours_per_week的系数等于0.04同时成立: "
<F test: F=array([[ 0.01940236]]), p=0.980784667777, df_denom=26042, df_num=2>
```

1 P-value小于0.05。拒绝education_num的系数等于0这个假设，即它的系数是显著的

2 P-value大于0.05。不能拒绝这两个假设同时成立

图 5-13

5.2.3 理解模型结果

通过上面的运算，可以得到逻辑回归模型的估计。但应该如何理解这得到的模型呢？比如模型中的参数表示什么？又比如当某一个变量变动时，它将如何影响最终的结果？下面来回答这两个问题。

为了数学推导上更加清晰，借鉴公式（5-10），我们假设有一个更简单的逻辑回归模型，如公式（5-17）。其中，P表示事件发生的概率，而x, z为自变量。

$$\ln {}^{P}/_{(1-P)} = ax + bz + c \tag{5-17}$$

1．事件的发生比：模型参数的意义

（1）在公式（5-17）中，令变量z不变，x从k增加到$k+1$，可以得到：

$$\ln odds(x = k) = \ln {}^{P}/_{(1-P)} = ak + bz + c$$

$$\ln odds(x = k+1) = \ln {}^{P}/_{(1-P)} = a(k+1) + bz + c$$

$$\ln {}^{odds(x = k)}/_{odds(x = k+1)} = a \tag{5-18}$$

（2）上面曾介绍过，$P/(1-P)$被称为发生比，记为 odds，表示某个事件发生与不发生的比率。将公式（5-18）稍做变换可以得到$odds(x = k)/odds(x = k+1) = \mathrm{e}^{a}$。这表示：在其他变量不变的情况下，当变量$x$增加 1 时，对应的发生比变为之前的$\mathrm{e}^{a}$倍[13]。

（3）将这一分析结果应用到 5.2.2 节里的模型：调用程序清单 5-5 中的第 9～15 行代码，可以得到如图 5-14 所示的结果。再次注意到，根据模型结果，年收入大于 5 万的发生比始终与每星期工作时间成正比：每增加 1 小时的工作时间，年收入大于 5 万的发生比就增加 4.07%。这与图 5-11 展示的事实不符，需要相应地修改模型。具体的办法将在第 7 章里再做讨论。

程序清单 5-5 理解模型结果——Statsmodels

```
1 | def interpretModel(re):
2 |     """
3 |     理解模型结果
4 |
5 |     参数
6 |     ----
7 |     re : BinaryResults，训练好的逻辑回归模型
8 |     """
9 |     conf = re.conf_int()
```

[13] 在统计上，$odds(x = k)/odds(x = k+1)$被称为机会比（odds ratio）。它常被用来度量：在不同组别间，某个事件的发生比是否存在明显差异。

```
conf['OR'] = re.params
# 计算各个变量对事件发生比的影响
# conf 里面的 3 列，分别对应着估计值的下界、上界和估计值本身
conf.columns = ['2.5%', '97.5%', 'OR']
print "各个变量对事件发生比的影响："
print np.exp(conf)
# 计算各个变量的边际效应
print "各个变量的边际效应："
print re.get_margeff(at="overall").summary()
```

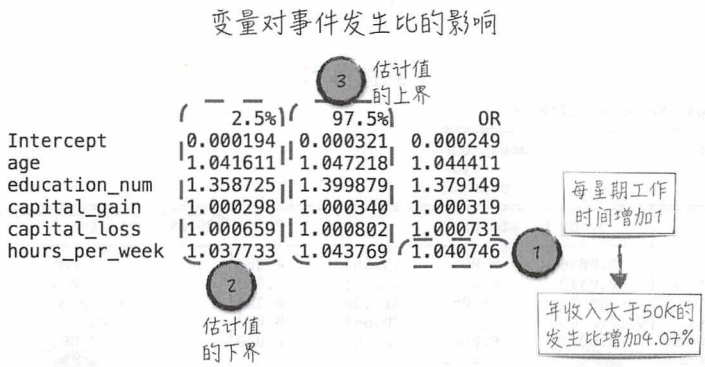


图 5-14

2．边际效应：变量变动对概率的影响

（1）在大多数情况下，相比于发生比，我们更关心变量对最终结果的影响。具体到逻辑回归，就是变量对事件发生概率的影响：当某个变量增加 1 时，事件发生的概率将如何变化？在学术上，这被称为变量的边际效应。从数学的角度来看，某个变量的边际效应等于被预测量对这个变量的导数。

（2）对公式（5-17）的两边分别求x的导数，得到公式（5-19）。

$$\frac{1}{P}\frac{\partial P}{\partial x} + \frac{1}{1-P}\frac{\partial P}{\partial x} = a$$

$$\frac{\partial P}{\partial x} = aP(1-P) \tag{5-19}$$

（3）公式（5-19）表示：当x增加 1 时，事件发生的概率将增加$aP(1-P)$。由于P表示事件发生的概率，因此对于不同数据点，参数a是恒定的，而 P通常不同。这导致对于某一个变量，各个数据点上的边际效应是不一样的[14]。这一点与线性回归模型不同，后者的边际效应是恒定的。

---

[14] 对于更一般的情况，我们对公式（5-10）的两边分别求$\boldsymbol{X}$的导数，可以得到：

$$\frac{\partial P}{\partial \boldsymbol{X}} = P(1-P)\boldsymbol{\beta}$$

在上面的公式里$\boldsymbol{\beta}$是恒定的，而P是变动的。所以同样可以得到：在逻辑回归模型里，变量的边际效应不恒定。

（4）由于变量的边际效应不恒定，因此在实际生产中，针对某一个变量，我们通常会先计算数据样本内所有数据点上的边际效应，再计算这些边际效应的平均值。而这个平均值将被认为是这个变量的边际效应。

（5）将这一分析结果应用到 5.2.2 节中的模型：调用程序清单 5-5 中第 18 行代码，可以得到如图 5-15 所示的结果。返回的结果里包含变量的边际效应，以及边际效应的置信区间。比如根据图 5-15，受教育年限（education_num）增加 1 时，年收入大于 5 万的概率将增加 0.0413；而且 95%的情况下，增加的概率将落在[0.040, 0.043]区间里。

变量的边际效应

```
        Logit Marginal Effects
=====================================
Dep. Variable:              label_code
Method:                            dydx
At:                             overall
==================================================================================
                 dy/dx    std err          z      P>|z|  [95.0% Conf. Int.]
----------------------------------------------------------------------------------
age              0.0056      0.000     33.563      0.000      0.005       0.006
education_num    0.0413      0.001     47.313      0.000      0.040       0.043
capital_gain   4.09e-05    1.3e-06     31.500      0.000   3.84e-05    4.34e-05
capital_loss  9.372e-05   4.54e-06     20.648      0.000   8.48e-05       0.000
hours_per_week   0.0051      0.000     28.167      0.000      0.005       0.005
==================================================================================
```

1 边际效应　　2 边际效应的置信区间

图 5-15

5.3 评估模型效果：孰优孰劣

逻辑回归模型训练好之后，我们就可以用它来对未知数据或者测试数据做预测了。正如公式（5-15）所示，逻辑回归模型的直接预测结果并非事件发生与否，而是事件发生的概率。换句话说，逻辑回归的结果并非我们想要的最终结果，而是需要进一步处理的中间结果。

如程序清单 5-6 中的第 6 行代码所示，使用 predict 函数对测试数据做预测。它返回的是年收入大于 5 万的概率（相应的事件发生的概率）。接着，需要制定概率到最终结果的转换规则。在示例的场景下，最终的结果是二元的，即年收入或大于、或不大于 5 万。如果模型预测，年收入大于 5 万的概率大于 0.5，这等价于年收入大于 5 万是最有可能发生的。因此，如第 12 行代码所示，我们的转换规则如下：当事件发生的概率（模型的预测值）大于 0.5 时，预测事件会发生。

如果将上面的话语翻译成数学公式，记号如下。

- $y_i = 1$表示居民i的年收入大于 5 万；$y_i = 0$表示居民i的年收入不大于 5 万；而$\hat{y}_i$是

y_i的预测值

- $\hat{P}(y_i = 1)$表示模型返回的预测值，即年收入大于 5 万的概率预测值。
- α表示事件发生的阈值，这里我们令$\alpha = 0.5$。

可以得到如下的公式：

$$\hat{y}_i = \begin{cases} 1, & \hat{P}(y_i = 1) > \alpha \\ 0, & \hat{P}(y_i = 1) \leqslant \alpha \end{cases} \tag{5-20}$$

显然改变阈值α的取值，会得到不一样的预测结果。比如降低阈值，会得到更多$\hat{y}_i = 1$的预测，如第 8～10 行代码所示。虽然阈值α能在很大程度上决定最终的预测结果，但它并不是逻辑回归模型本身的一部分。它与第 4 章中介绍的超参数类似，其取值依赖于业务场景和数据科学家的经验。下面将讨论如果根据实际情况选择合适的阈值α。

程序清单 5-6 预测结果

```
1  | def makePrediction(re, testSet, alpha=0.5):
2  |     """
3  |     使用训练好的模型对测试数据做预测
4  |     """
5  |     # 计算事件发生的概率
6  |     testSet["prob"] = re.predict(testSet)
7  |     print "事件发生概率（预测概率）大于 0.6 的数据个数："
8  |     print testSet[testSet["prob"] > 0.6].shape[0]  # 输出值为 576
9  |     print "事件发生概率（预测概率）大于 0.5 的数据个数："
10 |     print testSet[testSet["prob"] > 0.5].shape[0]  # 输出值为 834
11 |     # 根据预测的概率，得出最终的预测
12 |     testSet["pred"] = testSet.apply(lambda x: 1 if x["prob"] > alpha else 0,
                                axis=1)
13 |     return testSet
```

5.3.1 查准率与查全率

在讨论阈值α的选择方法之前，我们先来探讨：针对二元分类问题，应该如何正确评估一份预测结果的效果。这个也是选择阈值α的基础。

沿用上面的数学记号。如图 5-16 所示，图中标记为 1 的方块表示$\hat{y}_i = 0$，但$y_i = 1$的数据；标记为 3 的凹型方块表示$\hat{y}_i = 1$，但$y_i = 0$的数据；标记为 2 的方块表示$\hat{y}_i = 1$，且$y_i = 1$的数据[15]。而且这些图形的面积与对应数据的数据量成正比，比如$\hat{y}_i = 1$，且$y_i = 1$的数据个数越多，标记 2 的面积越大。

[15] 在实际应用中，对于一个二元分类问题，我们通常会更关注其中的某一类。比如在这个例子里，年收入大于 5 万是关注的焦点。在建模前的数据转换时，也常常将更关注的类别转换成 1。因此对预测结果进行评估时，关注点通常会聚焦于真实值或预测值等于 1 的情况。

很容易发现，图中标记为 2 的部分表示模型预测结果正确，而标记为 1 和 3 的部分则表示模型预测结果错误。

- 对于一份预测结果，一方面希望它能做到“精确”：当$\hat{y}_i = 1$时，有很大概率，真实值y_i就等于 1。这表现在图形上，就是标记 2 的面积很大，而标记 3 的面积很小。
- 另一方面也希望它能做到“全面”：对于几乎所有的$y_i = 1$，对应的预测值$\hat{y}_i$也等于 1。在图形上，这表示标记 2 的面积很大，而标记 1 的面积很小。

于是，对应地定义查准率（precision）和查全率（recall）这两个技术指标[16]来评估一份预测结果的效果。比较直观的定义如图 5-16 所示。

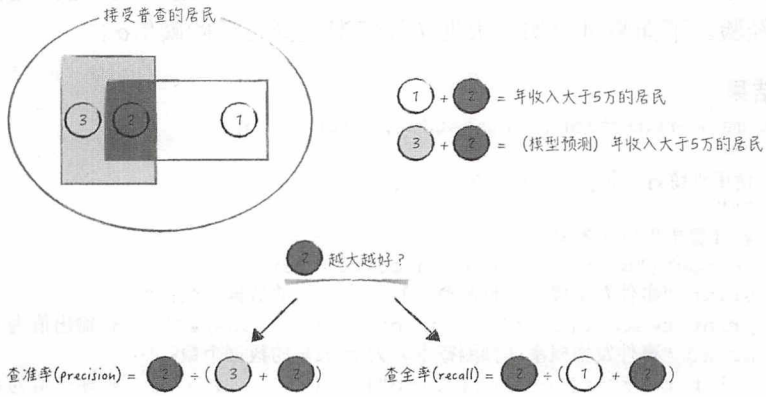


图 5-16

为了更加严谨，下面将从数学的角度给出这两个指标的严格定义。首先将数据按预测值和真实值分为 4 类，具体见表 5-2。

表 5-2[17]

		真实值	
		1	0
预测值	1	真阳性（true positive） TP （图 5-16 中的②部分）	伪阳性（false positive） FP （图 5-16 中的③部分）
	0	伪阴性（false negative） FN （图 5-16 中的①部分）	真阴性（true negative） TN

[16] 有的文献将查准率翻译为精确率，将查全率翻译为召回率。

[17] 在统计学里，伪阳性（false positive）又被称为第一型错误（type I error），而伪阴性（false negative）被称为第二型错误（type II error）。因此查准率通常被认为是衡量第一型错误的指标，而查全率衡量第二型错误的指标。

于是可以得到公式（5-21）：

$$Precision = {TP}/{(TP + FP)}$$
$$Recall = {TP}/{(TP + FN)} \tag{5-21}$$

公式（5-21）经过进一步的推导，可以得到这两个技术指标的概率定义，如公式（5-22）。从概率上来讲：预测值等于 1 时，真实值等于 1 的概率为查准率；真实值等于 1 时，预测值等于 1 的概率为查全率。

$$Precision = P(y_i = 1 \mid \hat{y}_i = 1)$$
$$Recall = P(\hat{y}_i = 1 \mid y_i = 1) \tag{5-22}$$

理想的情况是这两个指标都很高，但现实往往是残酷的。这两个指标通常存在着此消彼长的现象。比如对于一个逻辑回归模型，降低它的阈值α（公式（5-20）），往往会提高它的查全率，但同时会降低它的查准率，反之依然。整个过程的直观图像如图 5-17 所示。

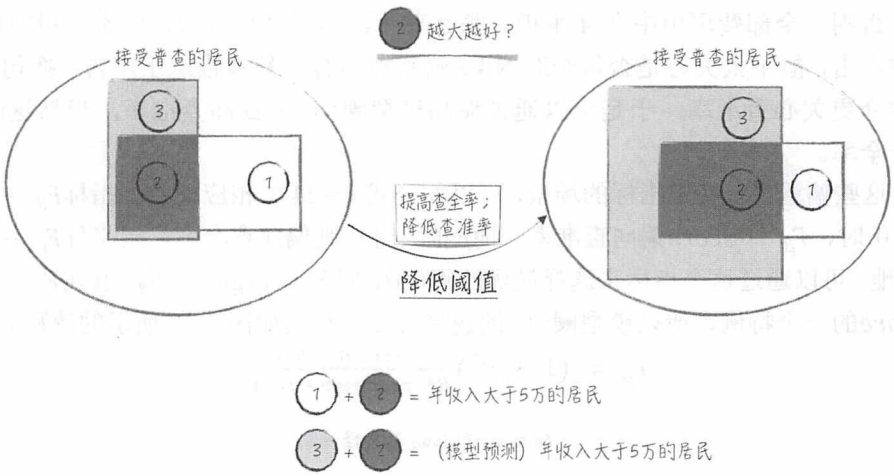


图 5-17

既然这两个指标往往是成反比的，而且在很大程度上，受阈值α的控制。那么只拿其中的某一个指标去评估预测结果是不太合适的。比如在极端情况下，预测所有居民的年收入都大于 5 万，即$\hat{y}_i \equiv 1$。这时预测的查全率是 100%，但查准率肯定很低。这样的预测显然是没太大价值的。而两个指标同时使用，在实际应用时又不太方便。为了破解这个困局，在实践中，我们定义了新的指标$F_1 - score$去“综合”这两个指标。具体的定义如公式（5-23），从数学上来看，它其实是查准率与查全率的调和平均数。对于二元分类问题，$F_1 - score$综合考虑了预测结果的查准率和查全率，是一个比较好的评估指标。因此，通常可以根据$F_1 - score$来选择最优的阈值$\hat{\alpha}$，即$\hat{\alpha} = \mathrm{argmax}_{\alpha} F_1$

$$F_1 = 2 \Big/ \left(\frac{1}{precision} + \frac{1}{recall}\right) = 2 \times \frac{precision \cdot recall}{precision + recall} \tag{5-23}$$

其实从逻辑回归模型的角度来看，查准率与查全率的“相互矛盾”给了我们更多的调整空间。应用场景不同，我们对查准率和查全率的要求是不一样的。在有的场景中，关注的焦点是查全率。例如 5.1 节里的衣服推荐，电商平台关心的是那些对衣服感兴趣的客户，希望模型对这些客户的预测都正确；而那些对衣服不感兴趣的客户，即使模型结果有较大偏差，也是可以接受的。也就是说，电商平台重视查全率，但不太关心查准率。这时就可以调低模型阈值α，通过牺牲查准率来保证查全率。但在有的场景中，查准率才是重点。例如在实时竞价（RTB）广告行业，有 3 种参与者：需要在互联网上对产品做广告的商家，比如 Nike；广告投放中介（DSP）；广告位提供者，比如新浪网。Nike 将广告内容委托给广告投放中介 A，A 通过分析选定目标客户群。当目标客户访问新浪网时，A 向新浪网购买广告位并将 Nike 广告推送给他。如果该客户点击了 Nike 广告，Nike 会向投放中介 A 支付相应费用。否则，全部费用由中介 A 承担。那么对于广告投放中介 A，它希望投放的每条广告都会被点击，但不太关心是否每个对 Nike 感兴趣的客户都被推送了广告。换句话说，广告投放中介更关心查准率。于是可以通过调高模型阈值α来提高查准率，当然这时会牺牲一部分查全率。

对于这些偏重某一特定指标的场景，可以如公式（5-24），相应地定义指标$F_\beta - score$。当β靠近 0 时，$F_\beta - score$偏向查准率，而β很大时，则偏向查全率。定义好$F_\beta - score$之后，类似地，可以通过这个指标来选择最优的阈值$\hat{\alpha}$，即$\hat{\alpha} = \text{argmax}_\alpha F_\beta$。其实$F_1 - score$是$F_\beta - score$的一个特例，所以模型阈值$\alpha$的选择可以总结为如图 5-18 所示的流程。

$$F_\beta = (1 + \beta^2)\frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{5-24}$$

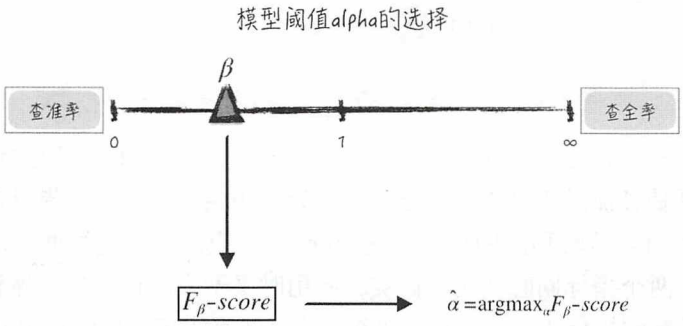


图 5-18

上面讨论了查准率、查全率以及$F_\beta - score$，它们都是很常用的二元分类问题评估指标。但遗憾的是，截止到本书编写时，第三方库 Statsmodels 并没有提供相应的计算函数，因此

需要自己动手实现。其实它们的计算过程并不复杂，具体的代码可参考程序清单 5-7。

程序清单 5-7 评估指标

```
import numpy as np

def evaluation(re):
    """
    计算预测结果的查准查全率以及 f1

    参数
    ----
    re : DataFrame，预测结果，里面包含两列：真实值‘lable_code’、预测值‘pred’
    """
    bins = np.array([0, 0.5, 1])
    label = re["label_code"]
    pred = re["pred"]
    tp, fp, fn, tn = np.histogram2d(label, pred, bins=bins)[0].flatten()
    precision = tp / (tp + fp)  # 0.951
    recall = tp / (tp + fn)  # 0.826
    f1 = 2 * precision * recall / (precision + recall)  # 0.884
    print "查准率: %3f, 查全率: %3f, f1: %3f" % (precision, recall, f1)
```

5.3.2 ROC 曲线与 AUC

讨论完查准查全率之后，我们想从另外的角度，再次探讨如何评估二元分类模型。仔细体会查全率的定义：$Recall = TP/(TP + FN)$，公式中的分母为实际上年收入大于 5 万的居民数，即数据中$y_i = 1$的数目。这个数字是固定的，与模型无关的，当然公式中的分子是受模型影响的。但是对于查准率：$Precision = TP/(TP + FP)$，公式中的分母为预测结果中，年收入大于 5 万的居民数，即$\hat{y}_i = 1$的数目。也就是说对于查准率，分子和分母都受模型的影响。因此当查准率变化时，很难分析清楚变化是来源于哪一部分。这影响了对模型效果的进一步分析。

为了解决这个问题，可以效仿查全率：分母固定，预测结果只影响评估指标的分子。于是，我们定义了真阳性率（True Positive Rate ，TPR）和伪阳性率（False Positive Rate，FPR）这两个指标。使用表 5-2 中定义的记号，这两个指标的具体定义如公式（5-25），相应的直观表示可参考图 5-20。

$$
\begin{aligned}
TPR &= {}^{TP}/_{(TP + FN)} \\
FPR &= {}^{FP}/_{(FP + TN)}
\end{aligned}
\tag{5-25}
$$

根据公式可以得到，真阳性率其实就是查全率。它表示针对年收入大于 5 万的居民有多大比例，模型对他们的预测结果也是年收入大于 5 万，它是衡量模型准确程度的指标。而伪阳性率表示针对年收入小于 5 万的居民有多大比例，模型对他们的预测结果却是年收入大于

5 万，它是衡量模型错误程度的指标。对于一份预测结果，显然希望它的真阳性率越高越好，而伪阳性率越低越好。

定义好真伪阳性率这两个评估指标后，我们考虑将它们图形化：以伪阳性率（False Positive Rate）为横轴、真阳性率（True Positive Rate）为纵轴画一个长度为 1 的正方形，形成所谓的 ROC 空间。那么对于一份预测结果，根据相应的真伪阳性率，可以将它表示为 ROC 空间中的一点，如图 5-19 所示。

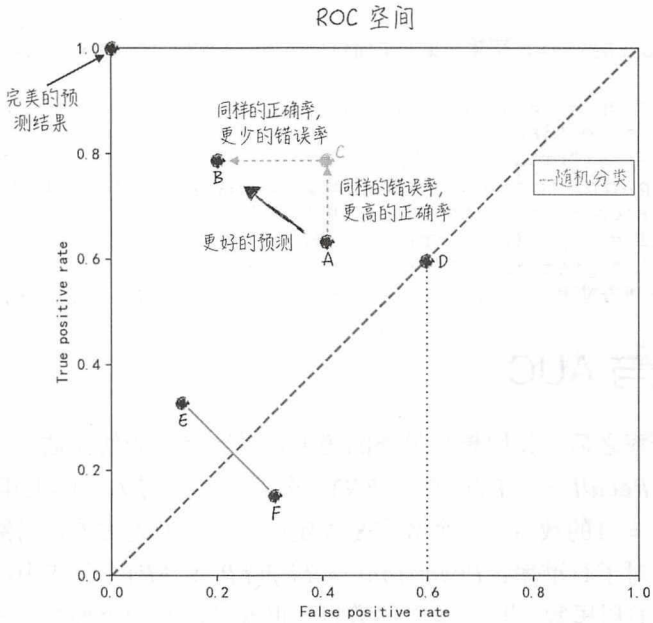


图 5-19

关于 ROC 空间，有以下几点值得注意。

- 在 ROC 空间里，离左上角越近的点预测准确率越高。比如图 5-19 中的 A、B 两点，B 点在 A 点的左上方，拥有更高的正确率和更低的错误率。因此它代表更好的预测结果。图中正方形左上角的点表示预测结果与真实情况完全一致，是一个完美的预测。
- 假设对于每一个数据点i，有 60%的概率预测$\hat{y}_i = 1$；有 40%的概率预测$\hat{y}_i = 0$。这完全是一个随机分类的预测结果。它的真伪阳性率都为 60%，对应着图中的 D 点。以此类推，可以得到，图中的虚线（即正方形的对角线）表示所有随机分类的预测结果。
- 显而易见，随机分类只是以一定概率猜测结果，没有任何的预测功能。这在理论上是最差的预测方式。但注意到图中的 F 点，它位于对角线的右下方，表示预测效果比随机分类更差。这是因为它的结果“搞反”了：只要依照 F 的结论做相反预测，得到新的预测

结果 E。E 的预测效果更好，并且好于随机分类。从图像上来看，E 是 F 以对角线为中线的镜像。

与查准查全率往往成反比不一样。真伪阳性率是正相关关系。比如对于一个逻辑回归模型，降低阈值时，这两个指标都会升高，整个过程如图 5-20 所示。直观一些理解，真阳性率是做预测得到的回报，而伪阳性率相当于所需付出的代价。就像生活一样，想要的回报越大，那么付出的代价也就越大。

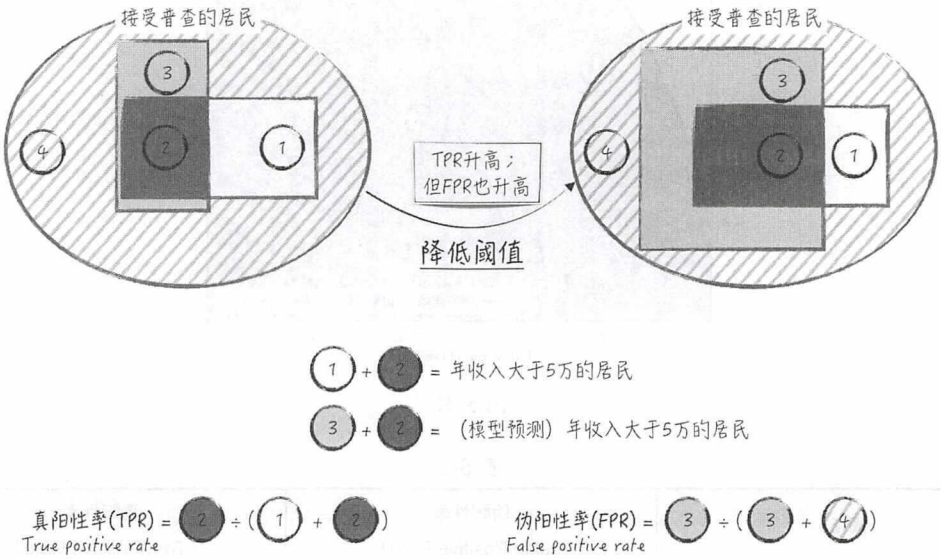


图 5-20

如果将模型的阈值从 0 缓慢增加到 1，并把每个阈值对应的预测结果记录在图上，就可以得到所谓的 ROC 曲线（接收者操作特征曲线，Receiver Operating Characteristic Curve），如图 5-21 所示，这里对应的是 5.2 节中的模型[18]。ROC 曲线下的面积，如图中的灰色部分，被称为 AUC。它是一个很重要的评估指标。简单来讲，AUC 越大，模型的预测效果越好。

我们可以更深入地来探讨 AUC 所代表的意义。在抽象的数学讨论之前，先来看一个简单的直观例子：数据集里一共有 4 个数据，分为被记为 1、2、3、4。这些数据按y_i的取值被分为两组：第一组包含数据 1、2，即$y_1 = y_2 = 1$；第二组包含数据 3、4，即$y_3 = y_4 = 0$。假设某个逻辑回归模型对它们的直接预测结果记为P_1, P_2, P_3, P_4，我们可以称这些结果为数据的得分。其中$P_1 = 0.8, P_2 = 0.4$，而$P_3 = 0.5, P_4 = 0.1$。所以模型阈值α与最终预测结果所

[18] 具体的实现请参考随书配套的代码/ch05-logit/example/roc_curve.py。

对应的真伪阳性率如表 5-3 所示。

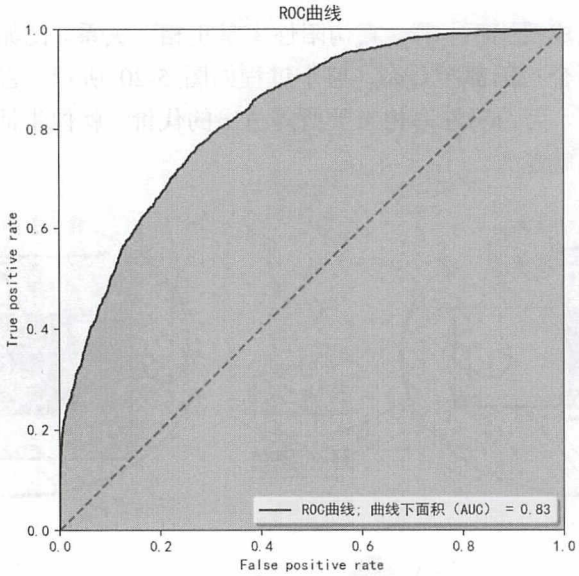


图 5-21

表 5-3

阈值α	伪阳性率（False Positive Rate）	真阳性率（True Positive Rate）
$\alpha \geqslant 0.8$	0	0
$0.5 \leqslant \alpha < 0.8$	0	0.5
$0.4 \leqslant \alpha < 0.5$	0.5	0.5
$0.1 \leqslant \alpha < 0.4$	0.5	1
$\alpha < 0.1$	1	1

将上面的结果表示在 ROC 空间里可以得到如图 5-22 所示的图形。其中 AUC 等于 0.75，这刚好等于第一组数据得分大于第二组数据得分的概率。

回到更一般的情况。5.3 节的开头提到，逻辑回归模型的直接预测结果是事件发生的概率，即$\hat{P}(y_i = 1)$。将这个预测值称为事件的得分。现在随机选取两个数据点k, l，其中$y_k = 1$，而$y_l = 0$。如果在预测结果中，k点的得分大于l点的得分，即$\hat{P}(y_k = 1) > \hat{P}(y_l = 1)$，这表示模型对这两个点的直接预测结果是正确的（当然，由于模型阈值的原因，最终预测结果不一定正确）。而 AUC 就表示这种情况发生的概率，即k点得分大于l点得分的概率。换句

话说，AUC 可以被视为模型预测正确的概率。用数学公式可以表示为：

$$AUC = P(\hat{P}(y_k = 1) > \hat{P}(y_l = 1)) \tag{5-26}$$

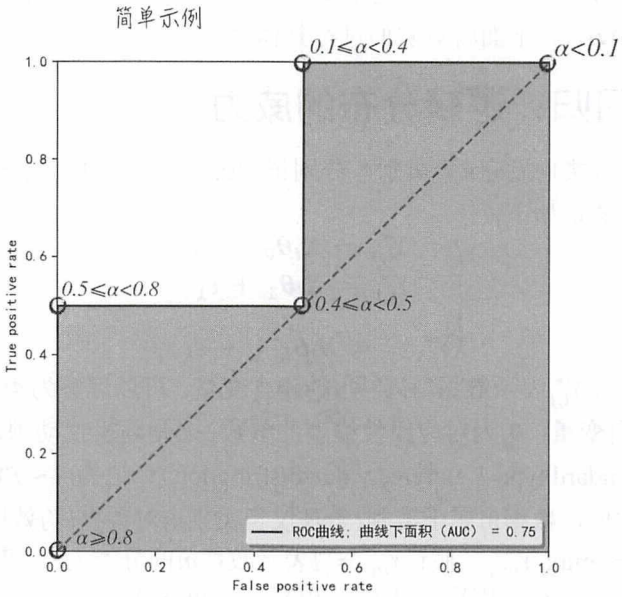


图 5-22

可以看到，AUC 与之前定义的查准查全率不同，它的取值不依赖模型阈值α，而完全取决于模型本身。因此它可以被认为是更加全面的模型评估指标。另外，虽然传统的 AUC 是定义在二元分类问题上的。但如果如公式（5-26）那样，从概率的角度理解它，可以帮助我们将这一指标推广到更多应用场景，比如大规模推荐系统。限于篇幅，具体细节这里就不做进一步展开了[19]。

5.4 多元分类问题：超越是与否

上面的讨论集中于二元分类问题。但在现实中，我们会碰到多元分类问题，即被预测量的取值多于两个，例如电商网站将用户分为普通用户、星级用户和疑似流失用户 3 类。对于这类问题，逻辑回归是否依然使用呢？答案是肯定的，但需要对模型做一些调整。常用的方法有两种：一种方法需要修改模型。具体地，它改变隐含变量的模型假设，推导出各个类别

[19] 有兴趣的读者可参考 *Advanced Analytics with Spark* 中的第 3 章。

新的概率分布公式。这种方法比较直接，也是逻辑回归模型所特有的，被称为多元逻辑回归模型（multinomial logit regression）。另一种方法则不需要改变现有的模型，而是将多元分类问题转换为多个二元分类问题。这种方法比较通用，几乎可以推广到所有的二元分类模型，被称为 One-vs-all（OvA）。下面将对它们进行具体介绍。

5.4.1 多元逻辑回归：逻辑分布的威力

假设面对的多元分类问题有k个类别，分别记为$0, 1, \ldots, k-1$。延续 5.1.2 节里的思路，修改隐含变量的模型假设如下：

$$\begin{cases} Y_{i,0}^* = \boldsymbol{X_i\theta_0} + \varepsilon_0 \\ Y_{i,1}^* = \boldsymbol{X_i\theta_1} + \varepsilon_1 \\ \cdots \\ Y_{i,k-1}^* = \boldsymbol{X_i\theta_{k-1}} + \varepsilon_{k-1} \end{cases} \tag{5-27}$$

在公式（5-27）里，$Y_{i,j}^*$表示数据i对类别j的隐含变量。可以理解为类别j对数据i的效用；$\boldsymbol{X_i}$表示数据i对应的自变量；$\boldsymbol{\theta_j}$为对应线性模型的参数；ε_j是随机扰动项，它服从标准的类型 1 极端值分布[20]（standard type-1 extreme value distribution），记为$\varepsilon_j \sim EV_1(0,1)$。

与二元的推导类似，数据m属于类别l当且仅当类别m对数据l的效用大于其他类别对数据l的效用，即$Y_{m,l}^* = \max_j Y_{m,j}^*$。假设$Y_m = l$表示数据$m$的分类为$l$。可以得到：

$$\begin{cases} P(Y_i = 0) = P(Y_{i,0}^* = \max_j Y_{i,j}^*) \\ P(Y_i = 1) = P(Y_{i,1}^* = \max_j Y_{i,j}^*) \\ \cdots \\ P(Y_i = k-1) = P(Y_{i,k-1}^* = \max_j Y_{i,j}^*) \end{cases} \tag{5-28}$$

经过一长串复杂的数学推导可以得到各个类别分布概率的具体表达式：

$$\begin{cases} P(Y_i = 1) = P(Y_i = 0)e^{\boldsymbol{X_i\beta_1}} = e^{\boldsymbol{X_i\beta_1}} \Big/ (1 + \sum_{j=1}^{k-1} e^{\boldsymbol{X_i\beta_j}}) \\ P(Y_i = 2) = P(Y_i = 0)e^{\boldsymbol{X_i\beta_2}} = e^{\boldsymbol{X_i\beta_2}} \Big/ (1 + \sum_{j=1}^{k-1} e^{\boldsymbol{X_i\beta_j}}) \\ \cdots \\ P(Y_i = 0) = 1 \Big/ (1 + \sum_{j=1}^{k-1} e^{\boldsymbol{X_i\beta_j}}) \end{cases} \tag{5-29}$$

---

[20] 极端值分布可视为正态分布的一种推广：它的概率密度函数和正态分布类似。从图像上来看，极端值分布的尾部比正态分布更厚一点。也就是说，极端值出现的概率更大一些。这使得它在金融领域应用很广，特别是在 2008 年金融危机之后，常被用来防范所谓的“黑天鹅”事件。

类型 1 极端值分布也被称为 Gumbel Distribution。它的概率密度函数如下：

$$EV_1(\mu, \beta) = \frac{1}{\beta} e^{(x-\mu)/\beta} e^{-e^{(x-\mu)/\beta}}$$

可以看到，多元逻辑回归与传统的逻辑回归很类似。其实当$k = 2$时，公式（5-29）就变回了逻辑回归的公式（5-9）。因此逻辑回归其实是多元逻辑回归的一个特例。有了各个类别的分布概率，可以类似地推导出参数的似然函数，再根据最大似然估计法得到参数的估计值[21]。对于多元逻辑回归模型，参数的似然函数如下，其中$1_{\{Y_i = j\}}$是 1 函数，当$Y_i = j$时，函数值等于 1，否则等于 0。

$$L = P(Y \mid \beta) = \prod_i \prod_{j=0}^{k-1} P(Y_i = j)^{1_{\{Y_i = j\}}}$$
$$\ln L = \sum_i \sum_{j=0}^{k-1} 1_{\{Y_i = j\}} \ln P(Y_i = j) \tag{5-30}$$

5.4.2 One-vs.-all：从二元到多元[22]

同样地，假设面对的多元分类问题有k个类别，分别记为$0, 1, \ldots, k-1$。从整体上看，现在的任务是将数据分为k类。但如果先盯住一个类，比如类别 1，那数据就只有两种状态了：属于类别 1 以及不属于类别 1。这就回到了我们很熟悉的二元分类问题。于是依此类推，为每一个类别建立一个分类器，将k元分类问题分解为k个二元分类问题。沿用上面的数学记号，$Y_i = j$表示数据i的分类为j，生成k个新的标签变量如下：

$$Y_{i,j}^{\sim} = \begin{cases} 1, & Y_i = j \\ 0, & Y_i \neq j \end{cases} \tag{5-31}$$

依次使用逻辑回归模型对数据$\{\boldsymbol{X}_i, Y_{i,j}^{\sim}\}$建模，得到相应的直接预测值，即在新的二元分类场景下，数据i属于类别j的概率，记为$\hat{P}_j(Y_{i,j}^{\sim} = 1)$：

$$\hat{P}_j(Y_{i,j}^{\sim} = 1) = 1 / (1 + e^{\boldsymbol{X}_i \hat{\boldsymbol{\varphi}}_j}) \tag{5-32}$$

其中$\hat{\boldsymbol{\varphi}}_j$为第$j$个逻辑回归模型参数的估计值。这里使用字母$\hat{\boldsymbol{\varphi}}_j$表示参数的估计值主要是为了和 5.4.1 节里的模型以示区别。

公式（5-32）中的$\hat{P}_j(Y_{i,j}^{\sim} = 1)$在某种程度上可以认为是最终标签$Y_i$等于$j$的概率。于是，选择这些值中最大值所对应的类别作为最终预测是合理的，从学术的角度来说就是选择产生最大置信度的分类模型所对应的标签，记$\hat{Y}_i$为Y_i的预测值。

$$\hat{Y}_i = j \Leftrightarrow \hat{P}_j(Y_{i,j}^{\sim} = 1) = \max_t \hat{P}_t(Y_{i,t}^{\sim} = 1)$$
$$\hat{Y}_i = \operatorname{argmax}_t \hat{P}_t(Y_{i,t}^{\sim} = 1) \tag{5-33}$$

对于多元分类问题，上面介绍的这两种算法的思想可以形象地表达为如图 5-23 所示的流程。

---

[21] 在实际应用中，由于多元逻辑回归的参数较多。为了避免过度拟合的问题，通常采用最大后验概率法（Maximum a Posteriori Estimation）来估计模型参数。

[22] 除了本书介绍的 one-vs.-all 方法（在有些文献里，此方法也被称为 one-vs.-rest，简称 OvA 或 OvR），还有一种类似的处理办法叫作 one-vs.-one。它将k元分类问题分解为$k(k-1)/2$个二元分类问题：将类别一一组合，再用二元分类器，比如逻辑回归，去解决这个二元分类问题。最后通过类似投票的方式，根据中间结果，选出最终的预测结果。

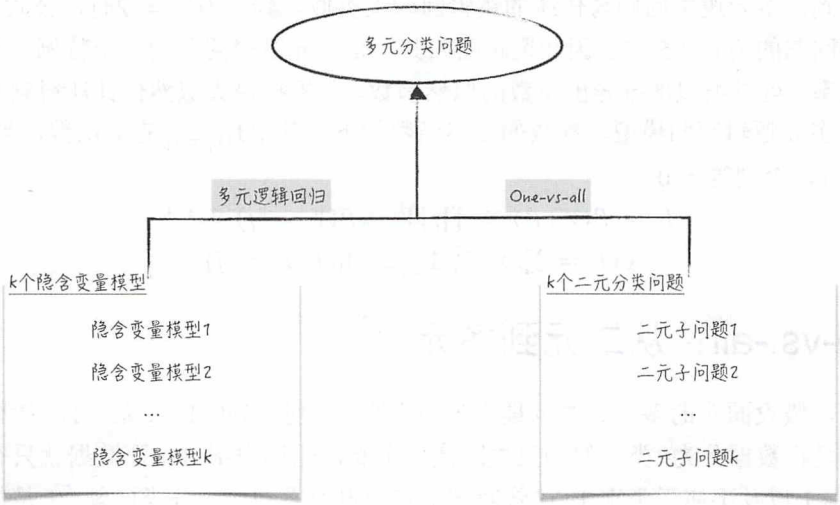


图 5-23

值得注意的是，one-vs.-all 这个方案很容易遇到非均衡数据集的问题，具体细节请参考 5.5 节。

5.4.3 模型实现

第三方算法库 scikit-learn 和 Statsmodels 都实现了这两种算法，调用起来非常方便。这里仅以 scikit-learn 为例，展示这两种模型的使用方法和效果差异。

如程序清单 5-8 中第 12、13 行代码所示[23]，通过 LogisticRegression 函数里的“multi_class”参数选择所用的模型方法。其中，“multinomial”表示多元逻辑回归；“ovr”表示 One-vs.-all 方法。

注意，第 12、13 行代码还对 LogisticRegression 函数里的“max_iter”和“random_state”参数做了修改。这些参数在模型理论推导时是不存在的，因为它们不是模型本身的参数，而是模型工程实现上的超参数。具体来讲，“max_iter”表示最大循环次数，而“random_state”表示随机状态。它们会被用来解决模型的最优化问题，也就是求解模型参数的估计值（通过求解模型损失函数的最小值来估算参数）。在实际应用中，这些超参数的取值能极大地影响模型参数的估计，从而影响模型的效果。它们的具体含义和选择技巧将在第 6 章中展开讨论。

程序清单 5-8 多元分类问题

```
1 | from sklearn.linear_model import LogisticRegression
2 |
3 | def multiLogit(data):
4 |     """
```

[23] 完整的实现请参考随书配套的代码/ch05-logit/example/multi_logit_example.py。

```
    使用逻辑回归对多元分类问题建模，并可视化结果
    """
    features = ["x1", "x2"]
    labels = "label"
    methods = ['multinomial', 'ovr']
    # 使用两种不同的方法对数据建模
    for i in range(len(methods)):
        model = LogisticRegression(multi_class=methods[i], solver='sag',
            max_iter=1000, random_state=42)
        model.fit(data[features], data[labels])
    ### # 略去后面的代码
```

如果将模型效果可视化，会得到如图 5-24 所示的图形。在图中，黑边圆圈表示原始数据，不同的颜色表示不同的类别。而 3 种不同的背景色表示模型的预测结果。如果圆圈的颜色和背景色相同，则表示预测正确；否则表示预测错误。

从图像上来看，两种方法的结果有差异，但区别并不大。从数学的角度来讲，这两种方法都有各自的优缺点，并不能说谁比谁更好。在理论上，这两者最大的不同之处如下。

- 多元逻辑回归将原始数据按类别分出一个个小的数据子集，然后在这些子集的基础上训练模型。因此模型参数的协方差矩阵在这些子集内是恒定的。
- One-vs.-all 的方法则是在个体的基础上建模，参数的协方差矩阵在上面的数据子集内也是不恒定的。

表现在结果上，多元逻辑回归会倾向于使所谓的决策边界（decision boundary）远离原始数据。所以在多元逻辑回归的结果图形里（见图 5-24a），更少的点会落在两种背景色的交界处。

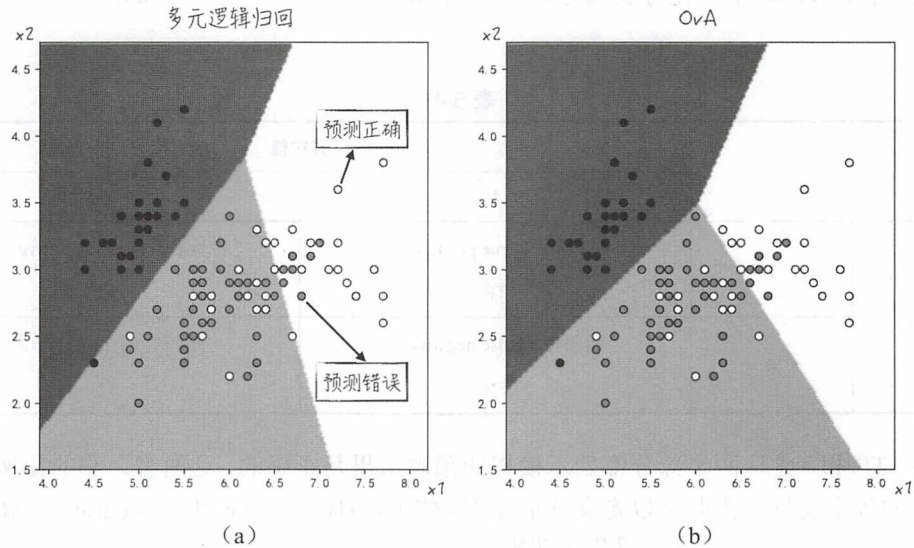


图 5-24

5.5 非均衡数据集

逻辑回归解决的是分类问题，而后者在实践中常常碰到非均衡数据集这个难题。非均衡数据集（imbalanced data）又称为非平衡数据集，指的是针对分类问题，数据集中各个类别所占比例并不平均。比如在网络广告行业，需要对用户是否点击网页上的广告进行建模。为了处理方便，我们记“点击广告”为类别 1，“不点击广告”为类别 0。因此这是一个二元分类问题。在训练模型的历史数据里有 1000 个数据点（1000 行），其中类别 1 的数据点只有 10 个，剩下的 990 个数据全部为类别 0。这就是一个非均衡数据集，类别之间的比例为 99:1。与二元分类问题类似，多元分类问题同样会面对非均衡数据集这个难题。不过在这个问题上，多元分类的处理的方案与二元的相似，因此为了表述简洁利于理解，下面的讨论将针对二元分类问题。

非均衡数据集在现实中是十分常见的。它给模型搭建带来了困难，如果不小心处理，会导致得到的模型结果毫无意义。在讨论这个话题之前，让我们稍稍离题一下，来看看所谓的准确度悖论（accuracy paradox）。

5.5.1 准确度悖论

回到 5.3.1 节，对于二元分类问题，模型的预测结果按准确与否可以分为如下 4 类，见表 5-4。

表 5-4

		真实值	
		1	0
预测值	1	真阳性（True positive） TP	伪阳性（False positive） FP
	0	伪阴性（False negative） FN	真阴性（True negative） TN

其中，TP 和 TN 这两个部分都表示模型的预测结果是正确的，这两者之和的比例越高，说明模型的效果越好。由此可以定义评估模型效果的指标——准确度（accurary，ACC）。

$$ACC = {(TP + TN)}/{(TP + FP + FN + TN)} \tag{5-34}$$

准确度这个指标看似很合理，但面对非均衡数据集时，这个指标会严重失真，甚至变得毫无意义。来看下面这个例子：数据集里有 1000 个数据点，其中 990 个为类别 0，而剩下的 10 个为类别 1，如图 5-25 所示。

准确度悖论：模型A比模型B更好？

模型A		真实值	
		1	0
预测值	1	0	0
	0	10	990

模型B		真实值	
		1	0
预测值	1	9	90
	0	1	900

$$ACC(A)=\frac{990+0}{990+10+0+0}=99\%$$

$$ACC(B)=\frac{900+9}{900+90+9+1}=90.9\%$$

图 5-25

模型 A 对所有数据的预测都是类别 0，因此这个模型其实并没有提供什么预测功能。但它的准确度却高达 99%。模型 B 的预测效果其实很不错：对于类别 1，10 个数据里有 9 个预测正确；而对于类别 0，990 个数据里有 900 个预测正确，但它的准确度只有 90.9%远低于模型 A。

这就是所谓的准确度悖论：面对非均衡数据集时，准确度这个评估指标会使模型严重偏向占比更多的类别，导致模型的预测功能失效。这也是在 5.3 节讨论模型评估时，我们并没有介绍准确度这个指标的原因。事实上，5.3.2 节里讨论的 AUC（曲线下面积）在面对非均衡数据集时，也能保持稳定，不会发生如准确度悖论这样的失真。

5.5.2 一个例子

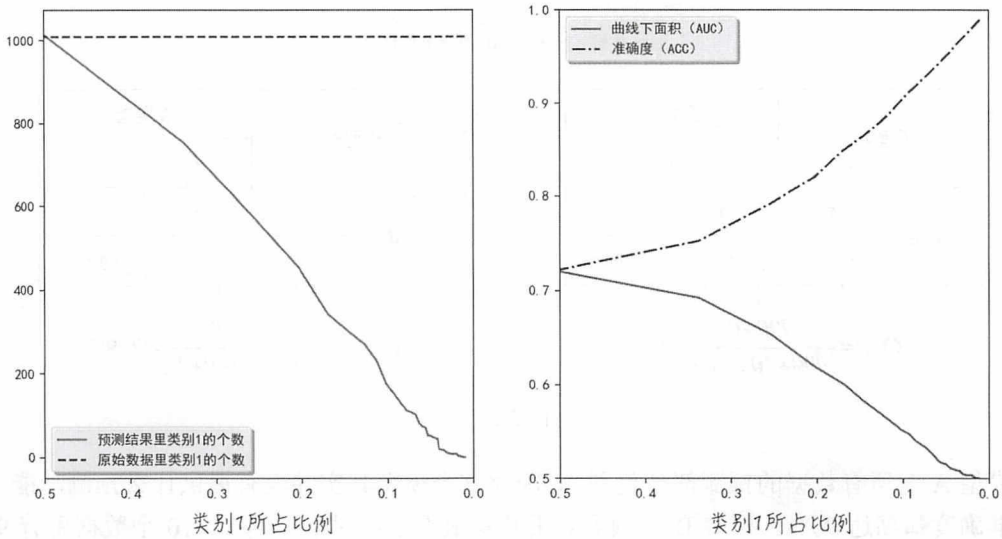
非均衡数据集除了会引起准确度悖论外，它对搭建模型有什么影响呢？下面通过一个简单的例子来说明这个问题。沿用公式（5-4）的思路，我们按公式（5-35）产生模型数据，其中变量y为因变量；x_1, x_2为自变量；ε为随机扰动项，它服从逻辑分布。

$$y=\begin{cases}1, & x_1-x_2+\varepsilon>0\\ 0, & else\end{cases} \tag{5-35}$$

由此可见，产生的模型数据完美地符合逻辑回归模型的假设。因此使用逻辑回归对数据建模，得到结果按理说应该非常好。但事实上，当数据集是均衡时，也就是说类别 1 所占比例大约为 0.5 时，模型效果是还不错。但当类别 1 所占比例接近 0 时，也就是数据集是非均衡时，模型的效果就很差了。虽然数据集里类别 1 的个数不变，但模型的预测结果几乎都是

书籍是人类进步的阶梯

类别 0，如图 5-26a 所示。正如 5.5.1 节中讨论的那样，ACC 这个指标在非均衡数据集里会失真，而 AUC 则可以保持稳定，能正确衡量模型的好坏，如图 5-26b 所示。



（a）　　（b）

图 5-26[24]

上面的例子从直观上展示了非均衡数据集对搭建模型的影响。那么造成这种结果的原因是什么呢？正如 5.1.4 节和 5.1.5 节讨论的那样，逻辑回归参数的估算公式如下：（惩罚项并不影响这里的讨论，因此我们在此省略掉惩罚项。）

$$h(\boldsymbol{X}_i) = {}^{1}/_{(1 + \mathrm{e}^{-\boldsymbol{X}_i\boldsymbol{\beta}})}$$

$$\widehat{\boldsymbol{\beta}} = \operatorname{argmin}_{\boldsymbol{\beta}} \sum_i -y_i \ln h(\boldsymbol{X}_i) - (1 - y_i)\ln[1 - h(\boldsymbol{X}_i)] \tag{5-36}$$

在这个公式里，每个数据点的权重都是一样的，都为 1。也就是说，模型对于类别 1 所承受的损失为：$-y_i \ln h(\boldsymbol{X}_i)$。这个值几乎等于模型对于类别 0 所承受的损失：$-(1 - y_i)\ln h(\boldsymbol{X}_i)$。如果某一类别的数据特别多，不妨假定为类别 0，那么在类别 1 某点的附件，极有可能存在大量的类别 0。在这种情况下，根据公式（5-36），模型会选择“牺牲”类别 1，从而导致预测结果几乎都为类别 0。

上面的结论并不只针对逻辑回归这个分类模型，对于其他分类模型，也同样成立。

[24] 具体的代码实现请参考随书配套的代码/ch05-logit/imbalanced_data.py。

书籍是人类进步的阶梯

5.5.3 解决方法

针对非均衡数据集，最常见也是最方便的解决方案是修改损失函数里不同类别的权重。以逻辑回归为例，将它的损失函数改写为公式（5-37）。

$$\hat{\boldsymbol{\beta}} = \operatorname{argmin}_{\boldsymbol{\beta}} \sum_i -w_1 y_i \ln h(\boldsymbol{X}_i) - w_0(1 - y_i)\ln[1 - h(\boldsymbol{X}_i)] \tag{5-37}$$

当类别 1 所占比例很少时，则增加w_1，也就是增加模型对于类别 1 所承受的损失，反之亦然。在大多数情况下，类别权重的选择原则是，类别权重等于类别所占比例的倒数[25]，如程序清单 5-9 中第 7、8 行代码所示。经过权重调整后，训练模型的数据集相当于回到了均衡状态。权重调整的代码非常简单，如第 10 行代码所示，通过“class_weight”参数调整各个类别的权重。事实上，“class_weight”也可以被赋值为“balanced”，即“class_weight='balanced'”，这时模型会自动调整各个类别的权重[26]。

程序清单 5-9 评估指标

```
1  from sklearn.linear_model import LogisticRegression
2
3  def balanceData(X, Y):
4      """
5      通过调整各个类别的比重，解决非均衡数据集的问题
6      """
7      positiveWeight = len(Y[Y>0]) / float(len(Y))
8      classWeight = {1: 1. / positiveWeight, 0: 1. / (1 - positiveWeight)}
9      # 为了消除惩罚项的干扰，将惩罚系数设为很大
10     model = LogisticRegression(class_weight=classWeight, C=1e4)
11     model.fit(X, Y.ravel())
12     pred = model.predict(X)
13     return pred
```

经过权重调整后，模型的结果如图 5-27 所示。在处理非均衡数据集时，调整权重后的模型会错误地将很多类别 0 的数据预测为类别 1。这与我们之前在 5.5.2 节里的分析是一致的：类别 1 的权重增加后，模型会因“刻意地珍惜”类别 1，而选择“牺牲”类别 0。尽管如此，调整之后的整体效果明显优于调整之前的（调整之后的 AUC 更大）。值得注意的是，ACC 和 AUC 这两个评估指标几乎相等，所以图形上它们两者重叠在了一起。

对于非均衡数据集，还有一些其他的解决方法，比如通过重新抽样（sampling），把多的类别变少或把少的类别变多。具体的细节请参考 *Learning from Imbalanced Data*[27]。

---

[25] 在没有惩罚项的情况下，类别权重的绝对大小并不影响模型结果，影响结果的是它们的相对大小。但有惩罚项的情况下，类别权重的绝对大小还会影响惩罚系数的选择。因此我们常常使类别权重之和等于 1。

[26] 当选择“class_weight='balanced'”时，各个类别的权重为$n_samples/(n_classes \times np.bincount(y))$。

[27] He H, Garcia E A. Learning from Imbalanced Data[M]. IEEE Educational Activities Department, 2009,21(9): 1263-1284.

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

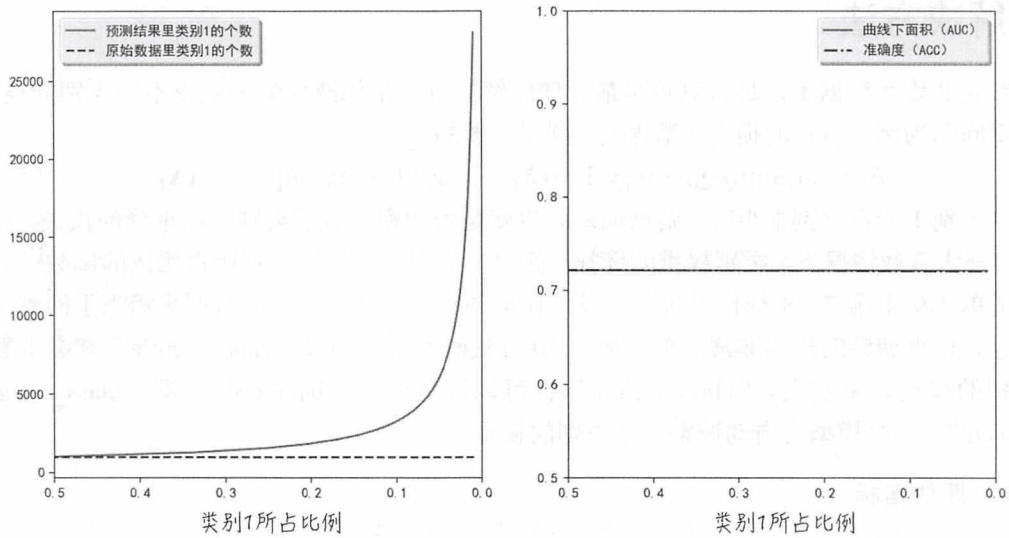


图 5-27

5.6 本章小结

本章介绍了在业界应用最为广泛的逻辑回归模型。我们从 3 个方面讨论了这个模型的细节。

在模型层面上，逻辑回归是被用来解决分类问题的。由于分类是一个非线性问题，所以建模的主要难点是如何把非线性问题转换为线性问题。从分解问题的角度入手：通过引入隐含变量，将问题划分为两个层次，一个是线性的隐含变量模型，另一个是基于隐含变量模型结果的非线性的变换。从图像的角度入手：逻辑回归通过非线性的空间变换，将原空间内非线性的分类问题转换为新空间内的线性问题。这是两种很常用的建模方法，在后面的章节里我们会发现，几乎所有的模型都是围绕着这两个思路展开的。

在模型评估层面，本章讨论了两类互相有关联的评估指标。对于分类问题的预测结果，可以定义相应的查准查全率，以及综合这两种效应的$F_\beta - score$。对于基于概率的分类模型，还可以绘制它的 ROC 曲线，以及计算曲线下面积 AUC。这些指标都有相应的概率解释。因此虽然它们都是定义在二元分类问题上的，但可以很自然地推广到多元分类问题以及其他场景，成为通用的评估手段。

最后，我们展示了如何使用逻辑回归处理多元分类问题。对于其他二元分类算法，也可以采用类似的处理办法将其推广为多元分类算法。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

由此可见，逻辑回归涉及的 3 个方面在机器学习领域是通用的。这也是印证了本章开头的一句话：“理解好逻辑回归的细节，就掌握了数据建模的精髓”。

当然到目前为止，我们讨论的都是模型比较宏观的知识。对于建模中比较细节的内容，会在后面的章节里一一展开。比如模型算法的工程实现，请参考第 6 章；又比如在建模时，应该如何选择特征等，将在第 7 章中详细讨论。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

第6章

工程实现：计算机是怎么算的

Ideas are cheap; execution is everything.
（创意不值钱，执行力才是关键。）

——*Chris Sacca*

6.1 算法思路：模拟滚动
6.2 数值求解：梯度下降法
6.3 上手实践：代码实现
6.4 更优化的算法：随机梯度下降法
6.5 本章小结

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

在之前的章节中，我们首先从场景入手分析数据；再通过数学变换把问题抽象为模型架构；最后借助 Python 的开源算法库，得到最终想要的模型，也就是参数已经被估计好的模型。但正如前面章节所展示的，模型参数的计算过程被开源算法库包了起来，只能通过相应的函数接口（API）调用它。从外部来看，模型参数的估计就像一个黑盒一样，根本不知道内部是如何运作的。第三方库这样设计的目的是为了将繁琐复杂的计算细节隐藏起来，方便用户使用。特别是在数据量较小时，使用传统模型的整个过程会非常简洁。但仅会调用算法库的 API 显然是不够的，特别是在一些前沿领域，比如最近很火的大数据和深度学习，理解模型参数的估算过程是非常关键的。

本章将讨论计算机是如何根据模型的数学公式，得到相应的参数估计值。用比较学术的语言来表述就是计算机解决最优化问题的算法。事实上，相应的算法有很多，不同的算法擅长解决的问题也不一样。但限于篇幅，本章只讨论的是其中最核心应用最广的算法——梯度下降法和随机梯度下降法。

这一部分的知识比较深奥，特别是对编程不太熟悉的读者可能会觉得整个内容生涩难懂。但幸运的是这一章相对来说比较独立，跳过它并不影响对模型本身的理解。因此，不感兴趣的读者可以选择先略过这一章，继续探索其他模型。等到需要时，比如想深入了解分布式机器学习（第 11 章）或深度学习（第 12、13 章）时，再回来阅读这部分内容。

6.1 算法思路：模拟滚动

如果抛开具体场景，从数学抽象的角度来看：不管是监督式还是非监督式，每个模型都有自己对应的损失函数，这个损失函数包含若干个未知的模型参数。比如针对回归问题，第 4 章中介绍了线性回归模型。它对应的损失函数如公式（6-1）所示，其中$\boldsymbol{X_i} = (x_{1,i}, x_{2,i}, \ldots, x_{k,i}, 1)$表示自变量向量，其中的 1 表示常数变量，$\boldsymbol{\beta} = (a_1, a_2, \ldots, a_k, a_{k+1})^{\mathrm{T}}$表示未知的模型参数。

$$L = \sum_{i=1}^{n}(y_i - \boldsymbol{X_i\beta})^2 \tag{6-1}$$

又比如针对分类问题，第 5 章中介绍了逻辑回归模型。它对应的损失函数如公式（6-2）所示，其中$y_i \in \{0, 1\}$表示数据所示的类别。

$$h(\boldsymbol{X_i}) = {}^{1}/_{(1 + \mathrm{e}^{-\boldsymbol{X_i\beta}})}$$

$$L = -\sum_i\{y_i \ln h(\boldsymbol{X_i}) + (1 - y_i)\ln[1 - h(\boldsymbol{X_i})]\} \tag{6-2}$$

前面已经讨论过，公式（6-1）是基于欧式距离的损失函数，在回归问题里很常用。而公式（6-2）的定义基于概率分布，是分类问题常用的损失函数，在学术上被称为交叉熵。

不论损失函数的具体形式如何，它的函数值都对应着模型的预测误差，因此这个值越小越好。仔细分析损失函数后会发现，由于模型使用的数据都是给定的，它的函数值完全取决

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

于模型的未知参数（以公式（6-1）和公式（6-2）为例，公式里的$\boldsymbol{X}_i, y_i$是不变的。L的取值完全取决于参数$\boldsymbol{\beta}$）。由此可以得到未知参数的估计原则：使得损失函数达到最小值。

下面我们就来看看如何求一个损失函数的最小值。为了表述方便，假设损失函数为$L(a,b) = (1/n)\sum_i(y_i - ax_i - b)^2$。对于数学比较熟悉的读者可能会发现，对于这个损失函数，可以快速求得参数估计值的解析表达式[1]，并由此得到具体的估计值。情况的确如此，但是这只是一个特例。在绝大多数情况下，损失函数是很复杂的，根本无法得到参数估计值的解析表达式，比如逻辑回归。因此需要寻找一种对大多数函数都适用的方法。

从另外一个角度来思考这个问题。L的函数图像如图 6-1 所示，可以把它想象成一个炒菜的圆底锅。在圆底锅的边上轻轻放下一颗鸡蛋，日常的生活经验告诉我们，不管圆底锅的形状如何，也无论鸡蛋的初始位置在哪里，鸡蛋最终会滚动到锅的最底部。对于求函数最小值，采取同样的思路：随机选取一个起始点，然后模拟鸡蛋滚动的样子，改变点的位置。最终使得选择的点到达函数图像的最低点，这时点所在的位置就是参数的估计值。，在数学上可以通过函数的导数来达到这个模拟效果，因为导数能揭示函数局部范围的图像性质，告诉我们“鸡蛋会往哪个方向滚”[2]。

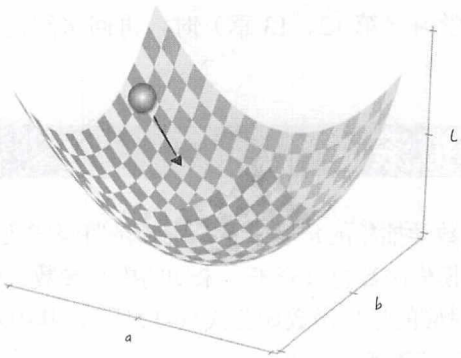


图 6-1[3]

[1] 对于一个处处可导的函数，函数最值的必要条件是导数值等于 0，即$\frac{\partial L}{\partial a} = 0, \frac{\partial L}{\partial b} = 0$。这两个公式对应的线性方程组刚好能解出参数估计值的表达式：

$$\hat{a} = \sum_i(x_i - \bar{x})(y_i - \bar{y}) \Big/ \sum_i(x_i - \bar{x})^2$$
$$\hat{b} = \bar{y} - \hat{a}\bar{x}$$

这组公式也被称为最小二乘法。

[2] 还有其他不同的求解最优化问题的方法，这些方法大多源自于模拟生活中的某个过程。比如模拟生物繁殖，得到遗传算法。又比如模拟钢铁冶炼的冷却过程，得到退火法。但这些算法都是工程实现方面的算法，跟正文中介绍的模型是两个不同的概念，比如对于线性回归模型，既可以用传统的梯度下降法求解，也可以用遗传算法来得到参数的估计值。

[3] 图片参考自 *Neural Networks and Deep Learning*。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

在学术上，这个方法被称为梯度下降法（gradient descent），下面将讨论这个算法的细节。

6.2 数值求解：梯度下降法

针对损失函数L，假设选取的初始点为a_0, b_0；现在将这两个点稍稍移动一点点，得到a_1, b_1。根据泰勒级数（Taylor series）[4]，暂时只考虑一阶导数[5]，可以得到公式（6-3），其中$\Delta a = a_1 - a_0, \Delta b = b_1 - b_0$：

$$\Delta L = L(a_1, b_1) - L(a_0, b_0) \approx \frac{\partial L}{\partial a}\Delta a + \frac{\partial L}{\partial b}\Delta b \tag{6-3}$$

如果令：

$$(\Delta a, \Delta b) = -\eta\left(\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b}\right) \tag{6-4}$$

其中$\eta > 0$，可以得到$\Delta L \approx -\eta[(\frac{\partial L}{\partial a})^2 + (\frac{\partial L}{\partial b})^2] \leqslant 0$。这说明如果按公式（6-4）移动参数，损失函数的函数值始终是下降的，这正是我们想要达到的效果。如果一直重复这种移动，数学上可以证明，损失函数能最终得到它的最小值，整个过程就像鸡蛋在圆底锅里滚动一样，于是可以得到参数的迭代公式（6-5）。

$$\begin{aligned} a_{k+1} &= a_k - \eta\frac{\partial L}{\partial a} \\ b_{k+1} &= b_k - \eta\frac{\partial L}{\partial b} \end{aligned} \tag{6-5}$$

也可以将公式（6-5）想象成在山坡上行走，为了下山：当发现脚下的路是下坡时，即$\partial L/\partial a < 0$，就应该顺着路的方向走一步；当发现脚下的路是上坡时，即$\partial L/\partial a > 0$，就应该逆着路的方向走一步。公式中的参数$\eta > 0$对应着步伐的长度。在学术上，$\eta$被称为学习速率（learning rate），是模型训练时一个很重要的超参数，能直接影响算法的正确性和效率：

- 一方面，参数η不能太大。因为从数学角度来讲，公式（6-3）是一阶泰勒展开，它是一个近似公式，只在学习速率很小，也就是$\Delta a, \Delta b$很小时才近似成立。从直观上来讲，如

---

[4] 回顾一下泰勒一阶展开式，假设$f(x_1, x_2, \ldots, x_n)$是一个一阶可导的函数，即$\frac{\partial^2 f}{\partial x_i \partial x_j}$都存在，则：

$$f(x_1, \ldots, x_n) = f(a_1, \ldots, a_n) + \sum_{i=1}^{n}\frac{\partial f(a_1, \ldots, a_n)}{\partial x_i}(x_i - a_i) + o\left(\sum_i |x_i - a_i|\right)$$

其中，$o(\sum_i |x_i - a_i|)$表示相对于$\sum_i |x_i - a_i|$的极小值，即：

$$\lim_{x \to a} o\left(\sum_i |x_i - a_i|\right) \Big/ \sum_i |x_i - a_i| = 0$$

因此在x很靠近a时，有$f(x) \approx f(a) + \sum_i \frac{\partial f(a)}{\partial x_i}(x_i - a_i)$。但是当$x$离$a$较远时，上述近似关系的误差就很大了。

[5] 如果考虑多阶导数，可以得到其他的最优化问题求解算法，比如使用两阶导数的共轭梯度法（Conjugate Gradient Method）等。这些算法对于特定问题可以更快地得到收敛解。但它们对损失函数的要求更多，计算复杂度也更高，并不适合神经网络和分布式机器学习，所以这里不做深入的探讨。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

果每次移动的步伐过大，容易发生来回摇摆的现象，无法到达最低点，影响算法的准确性，如图 6-2 所示。

- 另一方面，参数η也不宜过小。如果它的值太小，会导致每次迭代时，参数几乎不变化，使得算法的效率降低，需要很长时间才能到达最低点。

在数学上，向量$\nabla L = (\frac{\partial L}{\partial a}, \frac{\partial L}{\partial b})$被称为损失函数$L$的梯度。这也是公式（6-5）表示的算法被称为梯度下降法的原因。同时可以证明，函数的梯度正好是函数值下降得最快的方向，因此梯度下降法也是最高效的“下降”方式。

但梯度下降法，有一个很致命的问题。从理论上，它只能保证达到局部最低点，而非全局最低点，如图 6-3 所示。圆点从 1 位置开始往下滚，它只会留在局部最低点 3，而不是 3 旁边的全局最低点。发生这种情况与圆点的初始位置有关，如果将圆点的初始位置从 1 挪到 4，圆点就会最终到达函数的全局最低点。可以采用同样的思路解决梯度下降法的问题。具体步骤如下：首先随机产生多个初始参数集，即多组a_0, b_0；然后分别对每个初始参数集使用梯度下降法，直至函数值收敛于某个值；最后从这些值中找出最小值，这个找到的最小值被当作函数的最小值。

学习速率（learning rate）过大

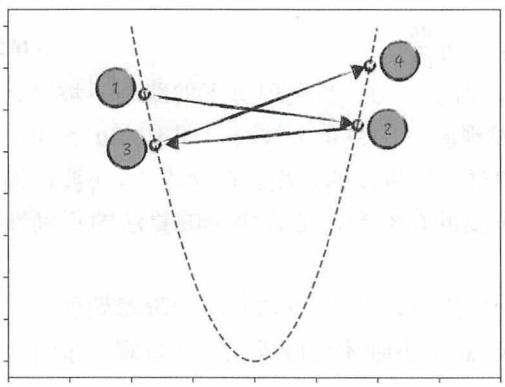


图 6-2

局部最低与全局最低

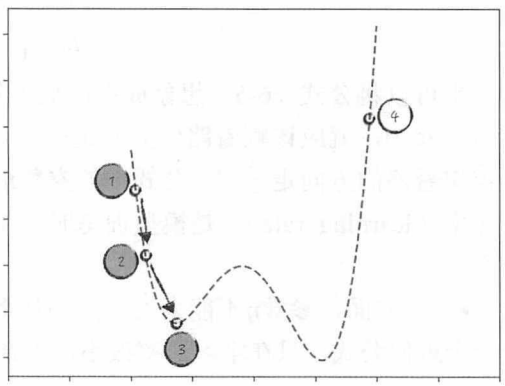


图 6-3

6.3 上手实践：代码实现

Python 是一种高级语言，它是建立在 C 语言基础上的编程语言。虽然 Python 容易理解和编写，但它的运算速度比较慢，并不适合进行大量的数值运算，因此从零开始实现高效的最优化算法是很困难的。但好在第三方模型库常常用其他底层语言实现高效的数值运算，并

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

将这些外部函数封装成 Python 自身的对象，方便用户使用（例如在之前章节里使用的 Numpy，它的运行方式就是这样）。站在这些基础函数的“肩膀”上，实现最优化算法就变得简单许多。下面将展示如何在 TensorFlow 的基础上实现梯度下降法。

6.3.1 TensorFlow 基础

TensorFlow 是由谷歌公司开源的人工智能系统，主要针对神经网络和深度学习领域。它对于传统算法的支持并没有像之前用到的 scikit-learn 那么成熟，但其独具特色的运算框架，使得 TensorFlow 有可能在未来成为机器学习领域里的一个标准化工具，如同 Android 系统对于智能手机一样。

TensorFlow 的运行方式就如同上面所表述的，将数值的密集运算放在 Python 之外进行。同时为了避免 Python 与外部程序切换时的开销，TensorFlow 并不单独地运行某一复杂计算。而是先利用 Python 将所有运算步骤转换一张计算图（computation graph），然后在将整张图传给外部函数进行运算。因此 TensorFlow 的编程通常可分为两部分：构造阶段和执行阶段。前者用于搭建计算图，并不做任何实质运算；后者则具体执行前者定义好的操作。整体的运行架构如图 6-4 所示，图中标注的计算图、变量、占位符将在后面一一讨论。

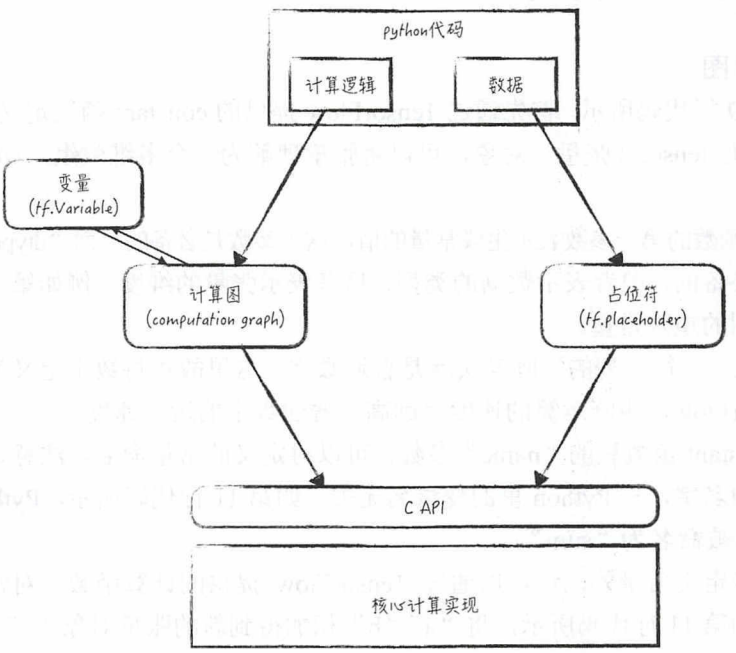


图 6-4

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

现在开始介绍 TensorFlow 的基础知识：如何构造计算图以及如何运行定义好的计算图。读者可以打开 Python shell 跟着一起练习。

程序清单 6-1 TensorFlow 基础

```
 1 | >>> import tensorflow as tf
 2 | >>>
 3 | >>> # 定义常量以及运算，形成运算图
 4 | >>> a = tf.constant(1, dtype=tf.float32, shape=[1, 1], name="a")
 5 | Tensor("a_5:0", shape=(1, 1), dtype=float32)
 6 | >>> b = tf.constant(2, dtype=tf.float32, shape=[1, 1], name="b")
 7 | Tensor("b_3:0", shape=(1, 1), dtype=float32)
 8 | >>> c = tf.constant(3, dtype=tf.float32, shape=[1, 1], name="c")
 9 | Tensor("c_3:0", shape=(1, 1), dtype=float32)
10 | >>> # s = a + b
11 | >>> s = tf.add(a, b, name="sum")
12 | Tensor("sum_3:0", shape=(1, 1), dtype=float32)
13 | >>> # re = c * s
14 | >>> re = tf.multiply(c, s, name="multiply")
15 | Tensor("multiply_3:0", shape=(1, 1), dtype=float32)
16 | >>>
17 | >>> # 启动与外部运算的会话，并通过‘run’函数计算定义好的图
18 | >>> sess = tf.Session()
19 | >>> print sess.run(s)
20 | [[3.]]
21 | >>> print sess.run(re)
22 | [[9.]]
```

1. 构造计算图

（1）如第 4～9 行代码所示，首先通过 TensorFlow 提供的 constant 函数定义常量“a”“b”“c”。这些常量都是 tensor（张量）对象，可以将张量理解为一个多维数组，和 Numpy 中的 array 对象类似。

（2）constant 函数的第一参数表示生成常量的值，这个参数是必备的。而“dtype”和“shape”这两个参数是非必备的，前者表示数据的类型，后者表示张量的维度。例如第 4 行代码表示生成一个一行一列的矩阵常量。

（3）在数学上，一行一列的矩阵其实就是普通数字。这里故意将数字定义为矩阵的形式是因为对于 TensorFlow，矩阵运算的速度远远高于普通数字的运算速度。

（4）通过 constant 函数里的“name”参数，可以对定义的常量命名。注意这个名字是常量在外部程序里的名字，与 Python 里的变量名无关。如第 11 行代码所示，Python 内的变量“s”在外部程序里被命名为“sum”。

（5）除了直接定义常量外，还可以通过 TensorFlow 提供的计算函数，对常量进行计算得到新的变量。如第 11 行代码所示，将“a”“b”相加得到新的张量对象“s”。

（6）第 4～15 行代码构造的计算图如图 6-5 所示。但需要注意的是，到目前为止，计算机并没有进行任何的计算，只是将 Python 代码翻译成外部程序可理解的操作，也就是上面

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

提到的构造阶段。

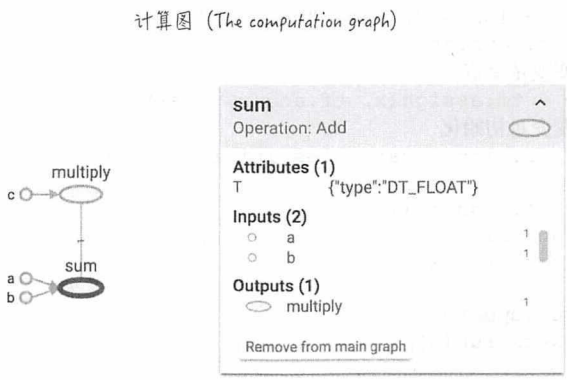


图 6-5

2. 执行定义好的计算图

（1）为了执行计算图，需要创建一个 session（会话）对象。如第 18 行代码所示。

（2）如第 19～21 行代码所示，通过 session 对象的 run 函数执行计算图上定义的运算。这个步骤也被称为数据的取回（fetches）。

（3）读者在运行代码时，可能会碰到如下的警告信息。不用担心，这些信息并不影响正常的使用。只是告诉你，可以下载 TensorFlow 的源码重新编译，以获得更快的运算速度。

程序清单 6-1 TensorFlow 基础

```
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use SSE4.1 instructions, but these are available on your
machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use SSE4.2 instructions, but these are available on your
machine and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use AVX instructions, but these are available on your machine
and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use AVX2 instructions, but these are available on your machine
and could speed up CPU computations.
W tensorflow/core/platform/cpu_feature_guard.cc:45] The TensorFlow library
wasn't compiled to use FMA instructions, but these are available on your machine
and could speed up CPU computations.
```

3. TensorFlow 提供的变量

除了上面的 constant（常量），TensorFlow 还提供另外的两种变量：Variable（可变变量）和 placeholder（占位符）。

书籍是人类进步的阶梯

程序清单 6-2 TensorFlow 变量

```
1  |  >>> # 定义可变变量，初始化为 0
2  |  >>> x = tf.Variable(0, name="counter")
3  |  >>> one = tf.constant(1)
4  |  >>> # 对可变变量赋值
5  |  >>> update = tf.assign(x, tf.add(x, one))
6  |  >>> # 对可变变量初始化
7  |  >>> init = tf.global_variables_initializer()
8  |  >>>
9  |  >>> sess = tf.Session()
10 |  >>> sess.run(init)
11 |  >>> print sess.run(x)
12 |  0
13 |  >>> sess.run(update)
14 |  >>> print sess.run(x)
15 |  1
16 |  >>> sess.run(update)
17 |  >>> print sess.run(x)
18 |  2
```

（1）如程序清单 6-2 中第 2 行代码所示，通过 Variable 函数创建可变变量，初始值为 0，名字为“counter”。

（2）如第 5 行代码所示，可以通过 assign 函数对可变变量再赋值，就像 Python 里的变量一样。注意 assign 函数改变的外部程序里的“counter”变量，而非 Python 里的“x”。

（3）与 TensorFlow 里的常量不同，对于可变变量，需要调用如第 7～10 行所示的代码，对可变变量进行真正的初始化 。

（4）如第 11～17 行代码所示，每运行一次“sess.run(update)”，“counter”变量就会被计算并重新赋值一次。整个过程如图 6-6 所示。

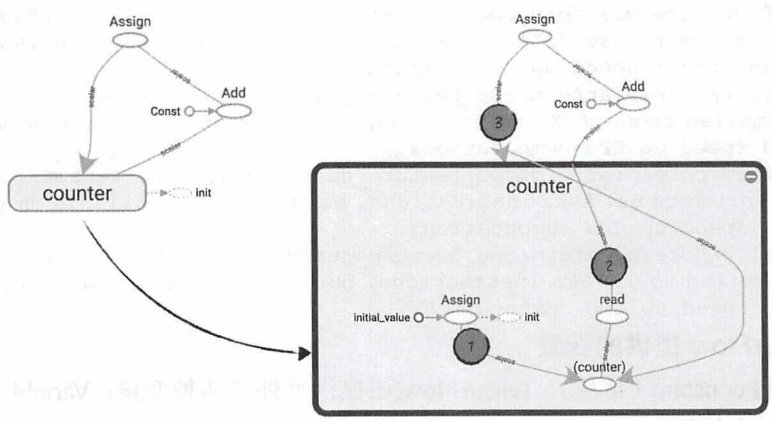


图 6-6

书籍是人类进步的阶梯

程序清单 6-2 TensorFlow 变量

```
19 | >>> import numpy as np
20 | >>>
21 | >>> # 定义占位符 mat1 和 mat2
22 | >>> mat1 = tf.placeholder(tf.float32, shape=[1, 3], name="mat2")
23 | >>> mat2 = tf.placeholder(tf.float32, shape=[3, 1], name="mat1")
24 | >>> output = tf.matmul(mat1, mat2)
25 | >>> # 将数据传给占位符，并进行计算
26 | >>> print sess.run([output], feed_dict={
27 | ...     mat1: np.array([1, 2, 3]).reshape(1, 3),
28 | ...     mat2: np.array([4, 5, 6]).reshape(3, 1)})
29 | [array([[ 32.]], dtype=float32)]
```

（5）占位符是一类很特殊的变量，可以把它理解为函数定义时的参数。通过它可以将数据从 Python 传给外部程序，并进行相应的计算。

（6）如第 22、23 行代码所示，定义占位符“mat1”和“mat2”。这两个变量并没有初始值，只是定义了所要数据的类型和格式，它等待着 Python 将数据传给它们。第 24 行代码中的“output”是这两个变量的乘积。

（7）如第 26～28 行代码所示，通过“feed_dict”这个参数，将数据传给相应的占位符，以便进行计算。这个步骤也被称为数据的供给（feeds）。

介绍完 TensorFlow 的基础知识，下面我们将以线性回归为例，展示如何实现梯度下降法。程序分为 3 步：首先随机产生呈线性关系的模型数据；然后在 TensorFlow 的基础上，定义线性回归模型，这包括模型中的自变量、因变量、模型参数、模型损失函数等；最后使用梯度下降法估计模型参数。具体的步骤如程序清单 6-3 所示[6]。

程序清单 6-3 程序结构

```
1 | def run():
2 |     """
3 |     程序入口
4 |     """
5 |     # dimension 表示自变量的个数，num 表示数据集里数据的个数。
6 |     dimension = 30
7 |     num = 10000
8 |     # 随机产生模型数据
9 |     X, Y = generateLinearData(dimension, num)
10 |    # 定义模型
11 |    model = createLinearModel(dimension)
12 |    # 使用梯度下降法，估计模型参数
13 |    gradientDescent(X, Y, model)
```

[6] 完整的实现请参数随书配套的代码/ch06-sgd/gradient_descent.py。

6.3.2 定义模型

线性回归模型的定义涉及 3 个方面（其实这 3 个方面对所有模型都是通用的）。

（1）模型数据里的自变量和应变量。如程序清单 6-4 中第 15～17 行代码所示，将模型所用到的自变量和应变量定义为占位符，方便在估计参数时传入所用的数据。值得注意的地方有两个：在定义占位符形状时，可以使用“None”值表示具体的数值未知，如“shape=[None, 1]”表示行数未知，列数为 1；正如前面提到的那样，将数值型应变量“y”定义成矩阵形式，这能极大地加快运算速度。

（2）模型参数。如第 19 行代码所示，将模型参数的估计值定义为可变变量。定义好模型参数的估计值，就可以由此计算模型的预测值“yPred”。

（3）损失函数。根据公式（6-1），定义损失函数等于预测值与真实值的差值平方和，具体的实现如第 22 行代码所示。

程序清单 6-4 定义模型

```
1  | def createLinearModel(dimension):
2  |     """
3  |     搭建模型，包括数据中的自变量，应变量和损失函数
4  |
5  |     参数
6  |     ----
7  |     dimension : int，自变量的个数
8  |
9  |     返回
10 |     ----
11 |     model : dict，里面包含模型的参数，损失函数，自变量，应变量
12 |     """
13 |     np.random.seed(1024)
14 |     # 定义自变量和应变量
15 |     x = tf.placeholder(tf.float64, shape=[None, dimension], name='x')
16 |     ## 将被预测值写成矩阵形式，会极大加快速度
17 |     y = tf.placeholder(tf.float64, shape=[None, 1], name="y")
18 |     # 定义参数估计值和预测值
19 |     betaPred = tf.Variable(np.random.random([dimension, 1]))
20 |     yPred = tf.matmul(x, betaPred, name="y_pred")
21 |     # 定义损失函数
22 |     loss = tf.reduce_mean(tf.square(yPred - y))
23 |     model = {"loss_function": loss, "independent_variable": x,
24 |         "dependent_variable": y, "prediction": yPred, "model_params": betaPred}
25 |     return model
```

6.3.3 梯度下降

梯度下降算法的核心是模型参数的迭代公式(6-5)。而 TensorFlow 已经实现了这个算法，相应的类为 GradientDescentOptimizer。因此在这个基础上实现梯度下降法其实非常简单[7]：

（1）如程序清单 6-5 中的第 14、15 行代码所示，首先创建 GradientDescentOptimizer 对象，再通过 minimize 函数传入需要估计的损失函数。

（2）由于模型参数是可变变量，因此创建 session（会话）对象，并对可变变量进行初始化，如第 26～30 行代码所示。

（3）通过 session 对象的 run 函数，传入模型数据更新模型参数的估计值，如第 37～40 行代码所示。循环调用这个代码，就可以得到损失函数的极小值以及相应的参数估计值。

（4）对于梯度下降法的工程实现，常用到的超参数有 3 个：控制“下降步伐”的“learningRate”；控制最大迭代次数的“maxIter”（注意这个超参数就是 5.4.3 节里曾介绍过的“max_iter”）；判断是否收敛的“tol”。当循环次数大于“maxIter”或者损失函数变动值小于“tol”时，就停止迭代，如第 36～46 行代码所示。

程序清单 6-5 梯度下降法

```
1 | def gradientDescent(X, Y, model, learningRate=0.01, maxIter=10000, tol=1.e-6):
2 |     """
3 |     利用梯度下降法训练模型。
4 |
5 |     参数
6 |     ----
7 |     X : np.array, 自变量数据
8 |
9 |     Y : np.array, 因变量数据
10 |
11 |     model : dict, 里面包含模型的参数，损失函数，自变量，应变量。
12 |     """
13 |     # 确定最优化算法
14 |     method = tf.train.GradientDescentOptimizer(learning_rate=learningRate)
15 |     optimizer = method.minimize(model["loss_function"])
16 |     # 增加日志
17 |     tf.summary.scalar("loss_function", model["loss_function"])
18 |     tf.summary.histogram("params", model["model_params"])
19 |     tf.summary.scalar("first_param", tf.reduce_mean(model["model_params"][0]))
20 |     tf.summary.scalar("last_param", tf.reduce_mean(model["model_params"][-1]))
21 |     summary = tf.summary.merge_all()
22 |     # 在程序运行结束之后，运行如下命令，查看日志
23 |     # tensorboard --logdir logs/
```

[7] 限于篇幅，这里实现的 gradientDescent 函数只能保证损失函数达到极小值。若想要得到损失函数的最小值，可通过多次调用这个函数来实现，具体的代码这里就不做讨论了。

```
24 |     summaryWriter = createSummaryWriter("logs/gradient_descent")
25 |     # TensorFlow 开始运行
26 |     sess = tf.Session()
27 |     # 产生初始参数
28 |     init = tf.global_variables_initializer()
29 |     # 用之前产生的初始参数初始化模型
30 |     sess.run(init)
31 |     # 迭代梯度下降法
32 |     step = 0
33 |     prevLoss = np.inf
34 |     diff = np.inf
35 |     # 当损失函数的变动小于阈值或达到最大循环次数，则停止迭代
36 |     while (step < maxIter) & (diff > tol):
37 |         _, summaryStr, loss = sess.run(
38 |             [optimizer, summary, model["loss_function"]],
39 |             feed_dict={model["independent_variable"]: X,
40 |                 model["dependent_variable"]: Y})
41 |         # 将运行细节写入目录
42 |         summaryWriter.add_summary(summaryStr, step)
43 |         # 计算损失函数的变动
44 |         diff = abs(prevLoss - loss)
45 |         prevLoss = loss
46 |         step += 1
47 |     summaryWriter.close()
48 |
49 | def createSummaryWriter(logPath):
50 |     """
51 |     检查所给路径是否已存在，如果存在删除原有日志。并创建日志写入对象。
52 |     """
53 |     if tf.gfile.Exists(logPath):
54 |         tf.gfile.DeleteRecursively(logPath)
55 |     summaryWriter = tf.summary.FileWriter(logPath, graph=tf.get_default_graph())
56 |     return summaryWriter
```

6.3.4 分析运行细节

在进行梯度下降法计算的同时，需要将运行时的日志记录下来，方便事后分析。例如参数是否真的收敛，收敛速度是否过慢等。通过 TensorFlow 提供的 summary 类，可以很轻松地完成这个任务。

（1）对于单个数值型变量（也就是所谓的标量），比如损失函数值，使用 scalar 函数记录它的信息，如程序清单 6-5 中第 17 行代码所示。

（2）对于矩阵型变量，比如模型参数（它是一个列向量），使用 histogram 函数记录信息，如第 18 行代码所示。这行代码将记录所有参数估计值的分布情况。如果想追踪某一特定参数，需要先将特定参数转换成标量，再使用 scalar 函数记录信息。具体实现参考第 19、20 行代码。

（3）summary 类定义的操作和前面介绍的其他 TensorFlow 操作一样，只有在 session 对象调用它们时，才会真正运行。但手工一个个调用显然效率不高，所以使用 merge_all 函数将它们合并为一个操作，如第 21 行代码所示。然后通过 run 函数触发它的运行，如第 37～40 行代码所示。

（4）为了记录产生的日志，需要创建“tf.summary.FileWriter”对象，如第 55 行代码所示。然后在每次迭代梯度下降法时，将相应的日志写入文件，如第 42 行代码所示。

在程序运行完毕之后，使用“tensorboard --logdir [your path]”命令启动 TensorBoard 查看日志。可以在 SCALARS 这一栏查看记录的标量信息，如图 6-7 所示。图中显示的是第 17、19 行代码运行的结果，其中横轴为迭代次数。

图 6-7

也可以在 HISTOGRAMS 下查看模型参数估计值的分布情况，如图 6-8 所示。图中显示的是第 18 行代码运行的结果，其中纵轴为迭代次数；横轴为参数估计值；图中的每一条横线表示，在某某迭代次数时，参数估计值的直方图。

参数估计值的分布情况可以用另外一种方式来表示，如图 6-9 所示。图中的横轴为迭代次数。图中从上到下一共有 9 条曲线，分别表示参数估计值的最大值、93%分位数、84%分位数、69%分位数、50%分位数（也就是中位数）、31%分位数、16%分位数、7%分位数、最小值。如果这些参数估计值服从正态分布，则图中的曲线分布表示$maximum$、$\mu + 1.5\sigma$、$\mu + \sigma$、$\mu + 0.5\sigma$、μ、$\mu - 0.5\sigma$、$\mu - \sigma$、$\mu - 1.5\sigma$、$minimum$，其中μ表示参数估计值的平均值，σ表示标准差。

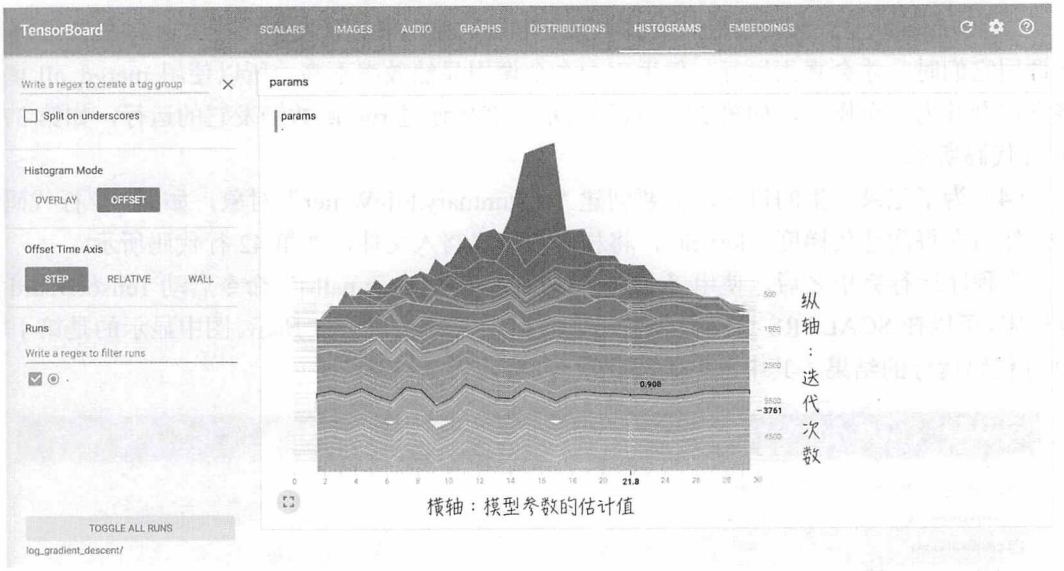


图 6-8

TensorFlow 之所以这样展示模型参数的估计值，是因为在神经网络和深度学习领域，模型的参数往往很多，有几百上千个。单个追踪的意义不大，需要从分布上把握参数估计值的收敛情况。

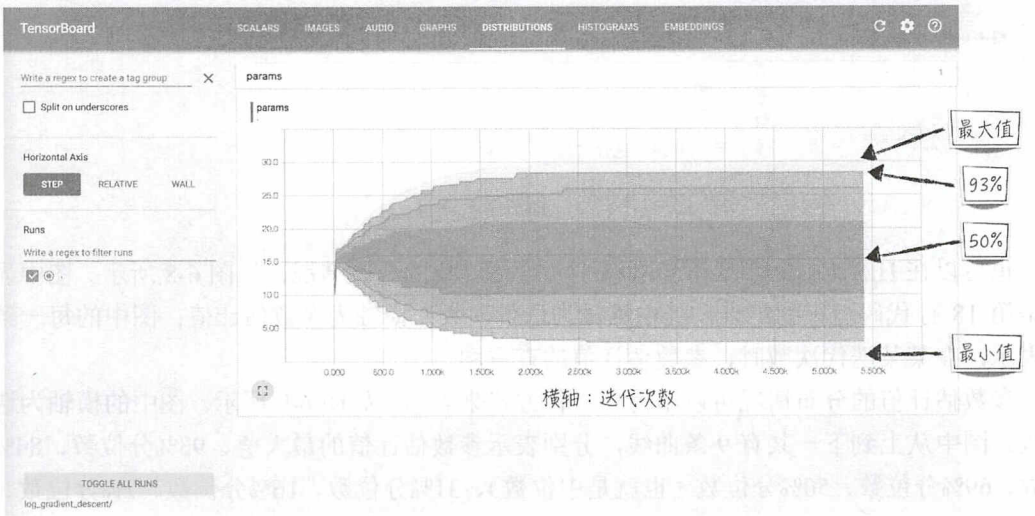


图 6-9

还可以在 GRAPHS 栏目下查看代码所构造的计算图，如图 6-10 所示。

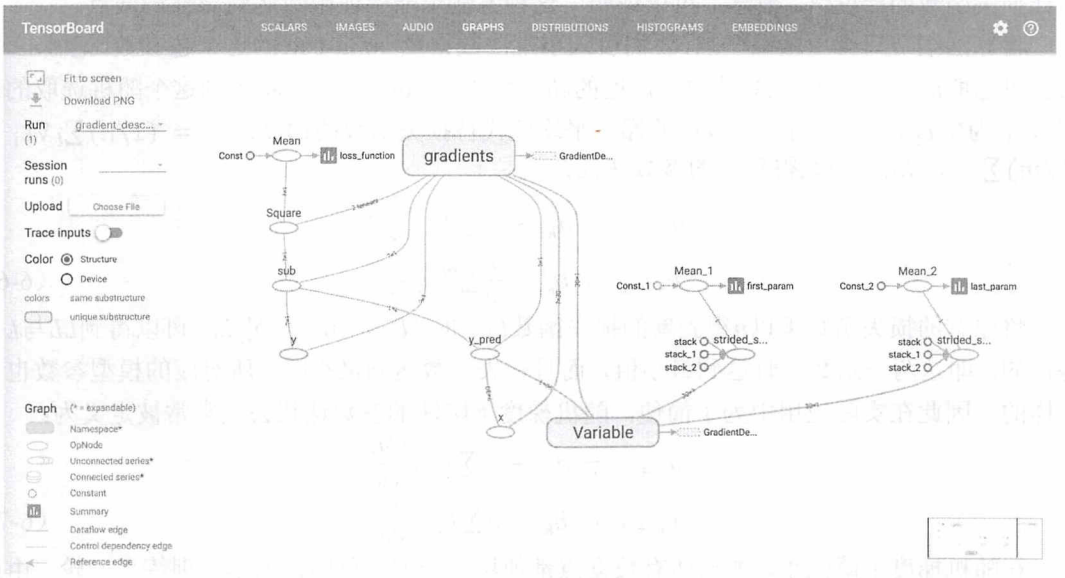


图 6-10

6.4 更优化的算法：随机梯度下降法

梯度下降法虽然在理论上是很美好的，但在实际应用中常常会碰到瓶颈。为了说明这个问题，我们令L_i表示模型在i点的损失，即$L_i = (y_i - ax_i - b)^2$，则损失函数L可以表示为：$L = (1/n)\sum_i L_i$。也就是说，模型的损失函数其实是模型在各个数据点的损失之和。这个结论对于模型是普遍适用的[8]。

计算损失函数L的梯度可以得到，$\nabla L = (1/n)\sum_i \nabla L_i$。也就是说，损失函数的梯度等于所有数据点处梯度的平均值。但是在实际应用中，模型用到数据集通常很大，计算这一次所有点的梯度之和，需要很长时间。为了加速这个计算过程，我们可以考虑使用随机梯度下降法（stochastic gradient descent）。

6.4.1 算法细节

随机梯度下降法的核心思想是：随机抽取若干个数据点，用这若干个点的梯度平均值去

[8] 对于解决回归问题的模型，这个结论是显然的。对于解决分类问题的模型，比如逻辑回归，只需对模型的似然函数做简单的数学变换（先求对数，再求相反数）就可以得到同样的结论。

估计损失函数的梯度[9]。数学上可以证明，这种方法也能保证函数达到局部最低点。

将算法细节写得更精确一点，我们引入小批次数据量（mini-batch）这个超参数，记做m。每次随机选取m个数据（5.4.3 节中所讨论的超参数“random_state”将控制这个随机选取的过程），记为$I_1, I_2, \ldots, I_m$。用这些数据的梯度平均值代替损失函数的梯度：$\nabla L = (1/n)\sum_i \nabla L_i \approx (1/m)\sum_{j=1}^{m} \nabla L_{I_j}$。由此得到新的参数迭代公式：

$$
\begin{aligned}
a_{k+1} &= a_k - \frac{\eta}{m}\sum_{j=1}^{m}\frac{\partial L_{I_j}}{\partial a} \\
b_{k+1} &= b_k - \frac{\eta}{m}\sum_{j=1}^{m}\frac{\partial L_{I_j}}{\partial b}
\end{aligned}
\tag{6-6}
$$

将原有的损失函数乘以n得到新的损失函数LL，即 $LL = nL = \sum_i L_i$。可以得到LL与L是等价的，即这两个函数同时达到最小值，而且损失函数达到最小时，所对应的模型参数也是一样的。因此在实际应用中为了简便，随机梯度下降法的参数迭代公式常常被定义为：

$$
\begin{aligned}
a_{k+1} &= a_k - \eta\sum_{j=1}^{m}\frac{\partial L_{I_j}}{\partial a} \\
b_{k+1} &= b_k - \eta\sum_{j=1}^{m}\frac{\partial L_{I_j}}{\partial b}
\end{aligned}
\tag{6-7}
$$

在随机梯度下降法里，如果所有数据点都使用了一遍，我们称为模型训练了一轮。由此引入另一个超参数——训练轮次“epoch”，表示所有数据将被用几遍。这个超参数与 6.3.3 节中介绍的“maxIter”类似，被用来控制随机梯度下降法里的循环次数（换句话说，就是公式（6-7）被迭代运算多少次）。

与梯度下降法相比，随机梯度下降法不但更高效，而且由于估算梯度时的随机性，能有效避免函数陷入局部最低的陷阱。当然它并不能完全避免这个问题，具体的解决方法与梯度下降法类似，这里就不再重复。

6.4.2 代码实现

随机梯度下降法的实现与梯度下降法类似，不同之处在于每次迭代更新模型参数时，需要“随机”选取一部分数据，具体的实现如程序清单 6-6 所示[10]。

（1）如 6.4.1 节中讨论的那样，在算法的工程实现上，我们引入“miniBatchFraction”和“epoch”两个超参数。如第 2 行代码所示，“miniBatchFraction=0.01”表示在默认情况下，计算一次公式（6-7）需要用到 1%的数据。换句话说，每次使用 1%的数据迭代模型，那么迭代 100 次才会使用全体数据 1 遍。“epoch=10000”表示在默认情况下，全体数据最多被使用 10 000 遍。这两个参数结合起来表示，模型最多被迭代$10\,000 \times 100$次。

---

[9] 这在数学上是完全合理的。从统计的角度来看，用所有数据点求平均值，并不比随机抽样的方法高明很多。与线性回归参数估计值类似，两个结果都是随机变量：它们都以真实梯度为期望，只是前者的置信区间更小。

[10] 这里省略了日志记录的代码，完整的实现请参考随书配套的代码/ch06-sgd/stochastic_gradient_descent.py。

（2）在具体的实现上，首先计算每次迭代使用的数据量，如第 25 行代码所示。然后按照数据的位置，依次选取不重叠的等量数据更新模型参数，如第 33～38 行代码所示。当所有数据都被使用过一遍后（由第 26 行代码中定义的“batchNum”控制），再次回到数据的初始位置并重复上面的过程。

（3）值得注意的是，与迭代模型只需要一部分数据不同，计算损失函数时需要使用全体数据，如第 40～43 行代码所示。

程序清单 6-6 随机梯度下降法

```
1  def stochasticGradientDescent(X, Y, model, learningRate=0.01,
2          miniBatchFraction=0.01, epoch=10000, tol=1.e-6):
3      """
4      利用随机梯度下降法训练模型。
5
6      参数
7      ----
8      X : np.array, 自变量数据
9
10     Y : np.array, 因变量数据
11
12     model : dict, 里面包含模型的参数，损失函数，自变量，应变量
13     """
14     # 确定最优化算法
15     method = tf.train.GradientDescentOptimizer(learning_rate=learningRate)
16     optimizer = method.minimize(model["loss_function"])
17     # TensorFlow 开始运行
18     sess = tf.Session()
19     # 产生初始参数
20     init = tf.global_variables_initializer()
21     # 用之前产生的初始参数初始化模型
22     sess.run(init)
23     # 迭代梯度下降法
24     step = 0
25     batchSize = int(X.shape[0] * miniBatchFraction)
26     batchNum = int(math.ceil(1 / miniBatchFraction))
27     prevLoss = np.inf
28     diff = np.inf
29     # 当损失函数的变动小于阈值或达到最大训练轮次，则停止迭代
30     while (step < epoch) & (diff > tol):
31         for i in range(batchNum):
32             # 选取小批次训练数据
33             batchX = X[i * batchSize: (i + 1) * batchSize]
34             batchY = Y[i * batchSize: (i + 1) * batchSize]
35             # 迭代模型参数
36             sess.run([optimizer],
37                 feed_dict={model["independent_variable"]: batchX,
38                     model["dependent_variable"]: batchY})
39             # 计算损失函数
40             loss = sess.run(
```

```
                [model["loss_function"]],
                feed_dict={model["independent_variable"]: X,
                    model["dependent_variable"]: Y})
            # 计算损失函数的变动
            diff = abs(prevLoss - loss)
            prevLoss = loss
            if diff <= tol:
                break
        step += 1
```

6.4.3 两种算法比较

分别使用梯度下降法和随机梯度下降法对线性回归模型做估计。正如前面分析的那样，两种算法能得到几乎一样的估计结果，但随机梯度下降法可以使模型参数更快地收敛，如图 6-11 所示。

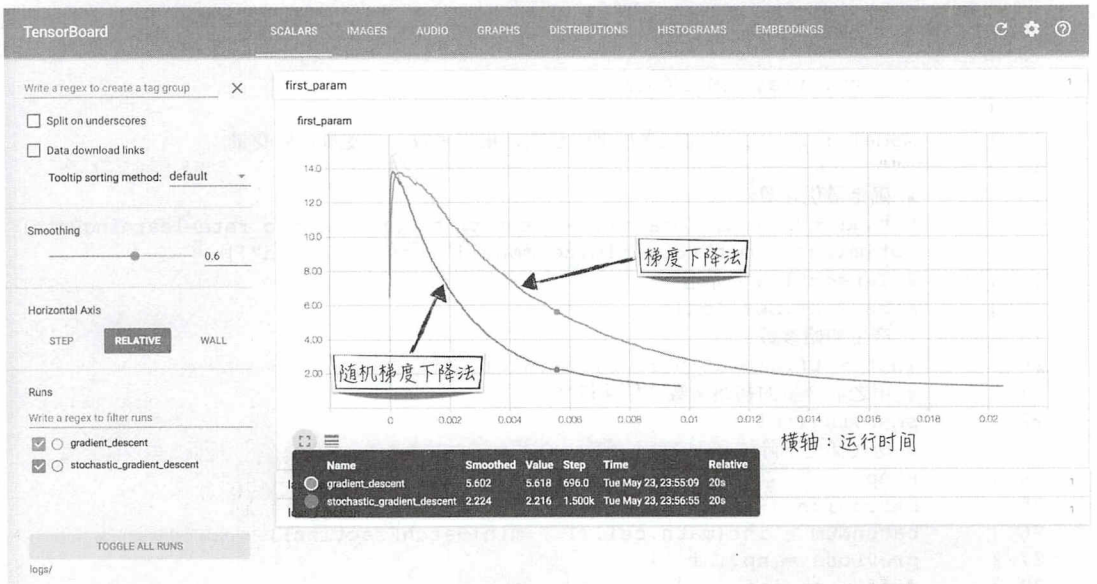


图 6-11

为了更细致地比较这两种算法的差异，如程序清单 6-7 所示[11]，在不同的数据量下，使用两种算法对同一线性回归模型做估计。可以发现数据量越大，随机梯度下降法的优势越明显，如图 6-12 所示。

[11] 完整的实现请参考随书配套的代码/ch06-sgd/gd_vs_sgd.py。

程序清单 6-7 梯度下降法 vs. 随机梯度下降法

```
def compareWithDiffSize():
    """
    在不同数据量下，使用两种算法对同一模型做估计
    """
    re = []
    dimension = 20
    model = createLinearModel(dimension)
    for i in range(1, 11):
        num = 10000 * i
        X, Y = generateLinearData(dimension, num)
        # 使用梯度下降法估计模型
        startTime = timeit.default_timer()
        gradientDescent(X, Y, model)
        endTime = timeit.default_timer()
        gdTime = endTime - startTime
        # 使用随机梯度下降法估计模型
        startTime = timeit.default_timer()
        stochasticGradientDescent(X, Y, model)
        endTime = timeit.default_timer()
        sgdTime = endTime - startTime
        re.append((num, gdTime, sgdTime))
    return re
```

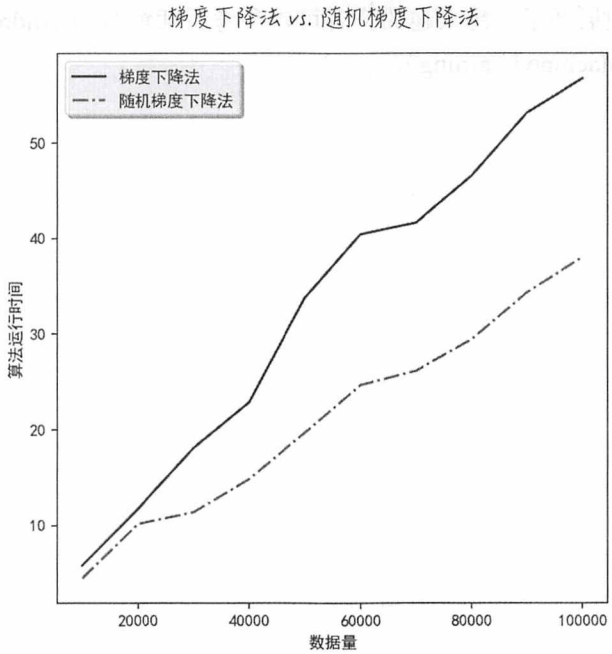


图 6-12

6.5 本章小结

对于模型的工程实现，核心问题是如何估计模型的参数，也就是如何求解最优化问题。本章以 TensorFlow 为基础工具，讨论了解决最优化问题的核心算法——随机梯度下降法（梯度下降法可视为随机梯度下降法的一个特例）。算法从给定的起始点开始，通过模拟小球滚动，得到损失函数的局部极小值。在这个基础上，多次随机选取起始点，重复上面的计算，最终求得函数的最小值。

随机梯度下降法的应用非常广。在神经网络和深度学习领域，常常使用基于它的反向传播算法（backpropagation）训练模型，具体的细节请参考第 12 章。在大数据分布式机器学习中，我们将随机梯度下降法分解到不同的机器上并行运算，提高算法的速度，这部分内容将在第 11 章中详细讨论。

当然对于某些特定的模型，有专门针对它们的最优化算法。比如针对支持向量学习机（SVM）的 SMO 算法（sequential minimal optimization）、针对生成式模型的最大期望算法（expectation maximization algorithm，EM）等。但这些算法的推导过程过于繁琐且超出了本章的范围，在此就不做展开了。感兴趣的读者可以参考后续章节和 Andrew Ng 在斯坦福大学的公开课《CS 229 Machine Learning》。

第7章

计量经济学的启示：他山之石

他山之石，可以攻玉。

——《诗经·小雅·鹤鸣》

7.1 定量与定性：变量的数学运算合理吗

7.2 定性变量的处理

7.3 定量变量的处理

7.4 显著性

7.5 多重共线性：多变量的烦恼

7.6 内生性：变化来自何处

7.7 本章小结

计量经济学（econometrics）是经济学中一个很重要的分支。它主要以数理统计学为基础，给经济学提供实证基础。计量经济学针对经济生活中的各种关系建立数学模型，并用建立好的模型对现实生活中的数据做分析，验证或推翻已有的经济学理论。它就像一把为经济学量身定做的直尺一样，使得经济学从以往的定性研究拓展到定量研究。第4章讨论的线性回归和第5章讨论的逻辑回归就是这门学科的核心模型。

经济学家凯恩斯[1]曾稍显自负地声称："经济学家和政治哲学家的思想，无论对错，都比通常所想更为有力。这个世界实际上就是被这些思想统治着的，很少例外。讲求实际的人们自以为能够与所有精神世界中的影响绝缘，到头来不过是某位已故经济学家的奴隶。"[2]经济学的影响力无疑是巨大的，因此需要尽可能地保证其做出的结论是准确的。而计量经济学是验证经济学理论的主要手段，自然就要求它的模型能做到如下几点。

- 模型是有效的，能抓住数据中的主要矛盾，做到从如山的数据堆中提取出简单且重要的关联关系。这样能避免两个常见的模型错误：一是找到一些由历史数据引起的虚假关联关系；二是将一些重要的关联关系舍弃掉。
- 模型是稳定的，得到的结果是放之四海而皆准的。换句话说，对于同一个问题，使用不同的历史数据集训练模型，得到结果是差不多的。这样模型就能摆脱对已知数据的依赖，得到的结论对未知数据也是有效的。
- 模型是可解释的，能做到知其然，并知其所以然。这样一方面能更好地保证模型的准确性；另一方面也能帮助我们更好地理解所处理的问题，进而制订改进措施。

以作者的观点，以上的3点正是机器学习领域所欠缺的，这也影响了机器学习在更大的范围内发挥作用。当然从客观上来说，相比于计量经济学，机器学习的模型更为复杂，做类似上面的分析十分困难。但是正如前面章节所强调的，复杂模型可以一层层分解为较为简单的子模型，比如线性回归。在这些子模型上借鉴计量经济学的方法做分析，可以帮助我们分析整体复杂模型的有效性和稳定性，也在一定程度上提供了模型的可解释性。虽然这样的做法在理论上有一些缺陷，结果不一定很完美，但也不失为一个解决方法。

本章将主要讨论计量经济学中对模型特征的处理：定量、定性特征各自的局限性和解决方法；模型特征使用时会遇到的显著性、多重共线性和内生性问题。其中后面两个问题在大数据时代愈发突出。

---

[1] 约翰·梅纳德·凯恩斯（John Maynard Keynes），英国经济学家。与传统的自由经济学思想不同，凯恩斯主张政府应积极扮演经济舵手的角色，透过财政与货币政策来对抗经济衰退乃至于经济萧条。这一主张成为19世纪20年代至19世纪30年代世界性经济萧条时的有效对策，也是19世纪50年代至19世纪60年代许多资本主义社会繁荣期经济政策的理论基础，因而被夸为资本主义的"救星"。他的理论学说被称为"凯恩斯学派"，在当今社会，特别是在我国影响力巨大（引自维基百科）。

[2] 出自《就业、利息和货币通论》（*The General Theory of Employment, Interest, and Money*）。

7.1 定量与定性：变量的数学运算合理吗

正如第 6 章里所讨论的那样，训练模型的本质是对训练数据做数学运算，并以此求得模型参数的估计值。所以我们需要保证两点：第一，训练数据能进行数学运算；第二，对变量所做的数学运算是合理的。对于第一点，通常只在一些特定的应用场景里才需要比较复杂的处理，比如自然语言处理、图像识别等。但对于第二点，几乎所有的场景都会遇到。这里将注重研究第二点“对变量所做的数学运算是合理的”，它表示的内涵是：对于变量，数字的运算有相应的实际意义，包括以下两个方面。

- 数字的大小关系。
- 数字的四则运算。

为了更好地讨论问题，将模型的自变量分类。正如前面章节所提到的，在模型里使用的变量可以分为两类：数值型变量和类别型变量。如图 7-1 所示。

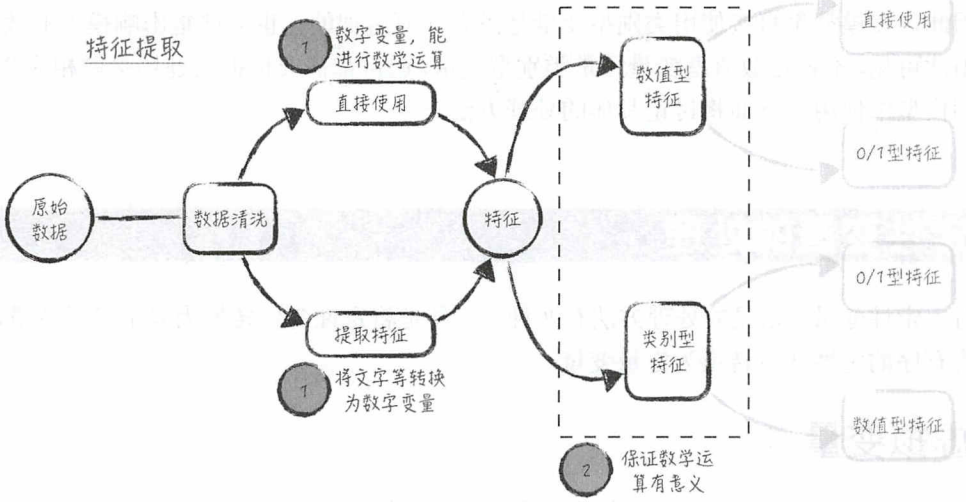


图 7-1

数值型变量，在学术上也被称为定量变量[3]（quantitative variable），如长度、收入、重量等。它们的数值表示具体的测量或计数。对于这类变量，数字间的不等关系是有实际意义

[3] 定量变量按是否连续可进一步细分为连续型变量和离散型变量。在一定区间内可以任意取值的变量叫连续型变量，比如人的身高、体重等；反之则是离散型变量，比如公司员工人数等。

的。比如对于收入，在数学上，100 小于 1000；在实际生活中，100 元也小于 1000 元。数字间的四则运算也同理，这里就不赘述了。当然由于数字的等于关系和四则运算，数值型变量常常隐含着边际效应恒定的假设，正如我们在第 5 章里讨论的那样。在某些场景下，这个隐含假设与现实不符，直接使用变量会影响模型的效果。

类别型变量，也被称为定性变量（categorical variable）。它并不是表示数量上的变化，而是表示类别。比如性别、省份、学历、产品等级等。这类变量的取值通常是用文字而非数字来表示。比如对于性别这个变量，可能的取值为男、女。因此要将文字变量转换为数字变量，并且保证对于转换之后的变量，数学运算是有意义的，这并不是一件容易的事情。通常针对一个类别型变量，我们会用一个数字去表示其中的一个类别，但这样的转换方法并不能满足要求：

- 对于有序的类别型变量，比如产品等级，0 表示合格、1 表示良好、2 表示优秀。这种情况下，0 小于 1 的确对应着合格等级次于良好等级，但数字间的四则运算就没有对应意义了。数学上 2 减 1 等于 1，但对于产品等级，优秀减去良好还等于良好吗？
- 对于无序的类别型变量，比如对于省份，0 表示北京、1 表示上海、2 表示深圳等。数字间的大小关系和四则运算都是没有实际意义的。

因此，在模型里直接使用类别型变量是没有任何道理的，也会严重影响模型的效果。

由此可见，不管是数值型变量还是类别型变量，通常都需要根据问题场景做相应变换后，再放到模型中使用。下面将讨论具体的处理方法。

7.2 定性变量的处理

对于定性变量，常见的处理方法有两种：一种是将定性变量转换为多个虚拟变量，另一种对将有序的定性变量转换为定量变量。

7.2.1 虚拟变量

正如前文中讨论的，直接对定性变量数字编码，得到的变量将无法进行有意义的数学运算。那么，相应的解决方法就是使得变换之后的变量不能直接做数学运算。

为了便于理解，我们先来看一个简单的例子：使用身高和性别对体重构建线性回归模型。性别是一个二元定性变量，可能的取值为男或女。用两个新生成的变量来取代性别，记为(x_1, x_2)。其中，$x_1 = 1$表示性别为男，$x_1 = 0$表示性别不为男；x_2类似，表示性别是否为女。在学术上，新生成的变量x_1, x_2被称为虚拟变量（dummy variable）。虚拟变量是一种特殊的离散型变量，可能的值只有两个——0 或 1，因此也被称为 0/1 变量。

书籍是人类进步的阶梯

用y表示体重，z表示身高，于是有：

$$y = ax_1 + bx_2 + cz + d + \varepsilon \tag{7-1}$$

注意到$x_1 + x_2 \equiv 1$，也就是变量x_1和变量x_2成线性关系。这会导致另外一个问题：多重共线性[4]（将在 7.5 节里讨论）。为了规避这个问题，我们对公式（7-1）做如下的数学变换，得到：

$$y = a(x_1 + x_2) + (b - a)x_2 + cz + d + \varepsilon$$

$$y = (b - a)x_2 + cz + (a + d) + \varepsilon \tag{7-2}$$

上面的数学转换可翻译为：首先选择性别男为基准类别，生成一维虚拟变量x_2，变量的含义与之前相同。这个变量前面的系数$b - a$表示性别女相对于性别男（基准类别）的体重差异。需要注意的是，针对二元定性变量，从表面上来看，直接对变量数字编码同虚拟变量效果一样。但这只是一个巧合而已，两种方法有本质的区别。

将上面的方法推广到n元定性变量（可能取值为n个的定性变量）。选择一个类别作为基准类别（对于基准类别的选择，请参考本章后面的 7.5.4 节），并生成$n-1$个虚拟变量，分别表示剩下的$n-1$个类别。在搭建模型时，用这$n-1$个新生成的虚拟变量代替原来的定性变量。具体过程如图 7-2 所示。

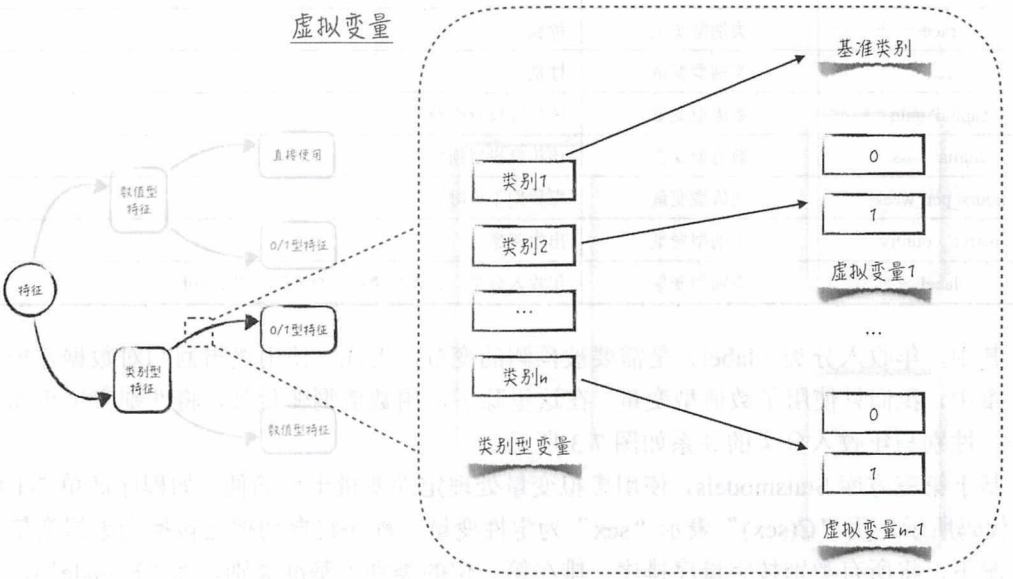


图 7-2

[4] 这个由虚拟变量引起的多重共线性问题在学术上被称为虚拟变量陷阱（dummy variable trap）。

书籍是人类进步的阶梯

7.2.2 上手实践：代码实现[5]

本节我们将讨论在代码层面上是如何将定性变量转换为虚拟变量。所用数据为美国个人收入的普查数据，在第5章中我们曾使用过此数据集进行建模。表7-1列举了数据集里的变量及其简单说明。

表7-1

变量名	变量类型	说明
age	数值型变量	年龄
workclass	类别型变量	工作类型，如公务员、私企职工等
fnlwgt	数值型变量	抽样权重。（普查时使用的变量，与建模分析无关）
education	类别型变量	学历，如本科、研究生等
education_num	数值型变量	受教育年限
martial-status	类别型变量	婚姻状况
occupation	类别型变量	所在行业
relationship	类别型变量	家庭角色，比如丈夫、妻子等
race	类别型变量	种族
sex	类别型变量	性别
capital_gain	数值型变量	该年度投资收益
capital_loss	数值型变量	该年度投资损失
hours_per_week	数值型变量	每星期工作时间
native_country	类别型变量	出生国家
label	类别型变量	年收入分类，分为两类：“>50K”和“<=50K”

其中，年收入分类（label）是需要被预测的变量，为此，使用逻辑回归对数据建模。在第5章中，我们只使用了数值型变量。在这里除了使用数值型变量外，将性别变量也加入模型中。性别与年收入分类的关系如图7-3所示。

基于第三方库Statsmodels，使用虚拟变量处理定性变量十分简便。如程序清单7-1中第7行代码所示，用“C(sex)”表示“sex”为定性变量。程序将自动把它转换为虚拟变量（默认情况下，将所有类别按字母序排序，排在第一位的类别为基准类别，即“Female”），加入模型中。

[5] 完整的实现请参考随书配套的代码/ch07-econometrics/categorical_variable.py。

书籍是人类进步的阶梯

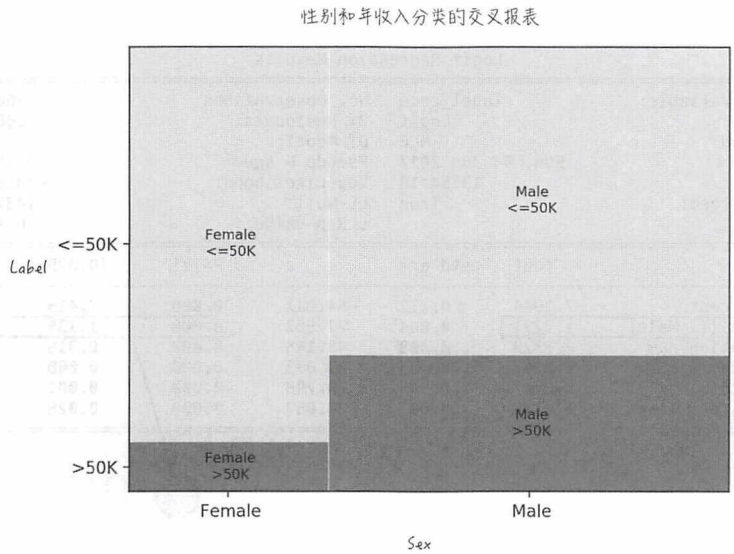


图 7-3

程序清单 7-1 定性变量

```
import statsmodels.api as sm

def trainModel(data):
    """
    搭建逻辑回归模型，并训练模型
    """
    formula = "label_code ~ C(sex) + education_num + capital_gain +
    capital_loss + hours_per_week"
    model = sm.Logit.from_formula(formula, data=data)
    re = model.fit()
    return re
```

模型的结果如图 7-4 所示，在同等条件下，男性年收入大于 50K（5 万美元）的概率更大，这一结论与图 7-3 是相符的。与数值型变量类似，可以计算各个虚拟变量对事件发生比的影响以及它们的边际效应。具体的实现可参考第 5.2.3 节，这里就不再重复了。

就模型本身而言，加入性别变量后，模型的预测效果有所提升[6]，如图 7-5 所示。

到目前为止，对于定性变量到虚拟变量的转换，我们只使用了最基础的默认方法。但在某些情况下，需要对变量转换做更精细的控制，比如将某几个类别同时定义为基准类别。这在代码层面应该怎么处理呢？为了说明这个问题，现在将另一个定性变量工作类型也加入模型中，得到的结果如图 7-6 所示。

[6] 在具体实现上，由于使用了交叉验证技术来评估模型，所以加入新变量并不一定会提升模型的预测效果。

书籍是人类进步的阶梯

```
                           Logit Regression Results
==============================================================================
Dep. Variable:             label_code   No. Observations:                26048
Model:                          Logit   Df Residuals:                    26042
Method:                           MLE   Df Model:                            5
Date:                Sun, 04 Jun 2017   Pseudo R-squ.:                  0.2582
Time:                        13:54:18   Log-Likelihood:                -10660.
converged:                       True   LL-Null:                       -14370.
                                        LLR p-value:                     0.000
==================================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------
Intercept         -7.1944      0.112    -64.011      0.000      -7.415      -6.974
C(sex)[T. Male]    1.2221      0.044     27.602      0.000       1.135       1.309
education_num      0.3329      0.008     43.145      0.000       0.318       0.348
capital_gain       0.0003   1.08e-05     31.073      0.000       0.000       0.000
capital_loss       0.0007    3.6e-05     20.798      0.000       0.001       0.001
hours_per_week     0.0308      0.001     20.857      0.000       0.028       0.034
==================================================================================
```

1 sex=Male的系数大于0

2 变量系数是显著的

图 7-4

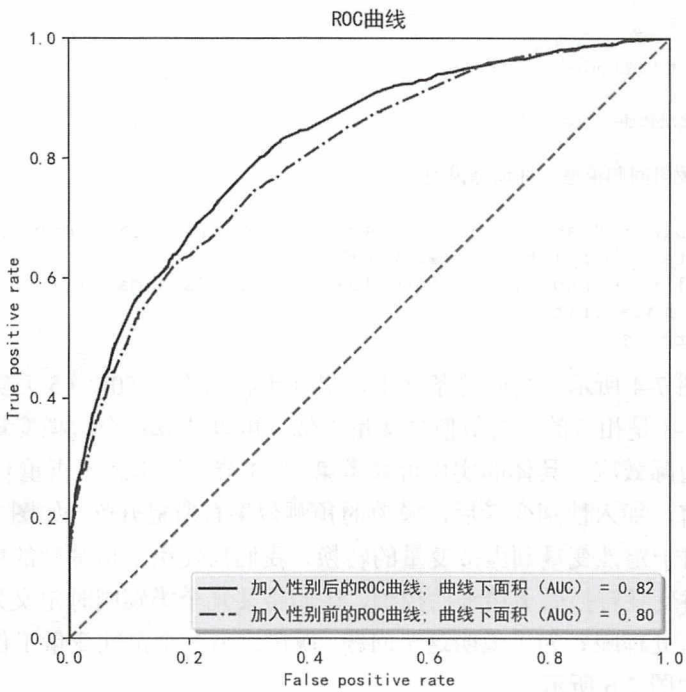


图 7-5

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

```
                           Logit Regression Results
==============================================================================
Dep. Variable:              label_code   No. Observations:                26048
Model:                           Logit   Df Residuals:                    26034
Method:                            MLE   Df Model:                           13
Date:                 Sun, 04 Jun 2017   Pseudo R-squ.:                  0.2643
Time:                         17:03:44   Log-Likelihood:                -10572.
converged:                        False   LL-Null:                       -14370.
                                          LLR p-value:                     0.000
=================================================================================================
                                      coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------
Intercept                        -7.5636      0.145    -52.163      0.000      -7.848      -7.279
C(workclass)[T. Federal-gov]      1.0642      0.132      8.076      0.000       0.806       1.323
C(workclass)[T. Local-gov]        0.5863      0.118      4.975      0.000       0.355       0.817
C(workclass)[T. Never-worked]   -21.7010   9.67e+04     -0.000      1.000   -1.9e+05     1.9e+05
C(workclass)[T. Private]          0.4907      0.102      4.829      0.000       0.292       0.690
C(workclass)[T. Self-emp-inc]     1.2769      0.129      9.898      0.000       1.024       1.530
C(workclass)[T. Self-emp-not-inc] 0.3536      0.117      3.032      0.002       0.125       0.582
C(workclass)[T. State-gov]        0.3371      0.131      2.583      0.010       0.081       0.593
C(workclass)[T. Without-pay]    -15.9531   2277.098     -0.007      0.994   -4478.983    4447.077
C(sex)[T. Male]                   1.2101      0.045     27.101      0.000       1.123       1.298
education_num                     0.3283      0.008     41.729      0.000       0.313       0.344
capital_gain                      0.0003   1.08e-05     30.763      0.000       0.000       0.000
capital_loss                      0.0007   3.62e-05     20.611      0.000       0.001       0.001
hours_per_week                    0.0288      0.002     18.946      0.000       0.026       0.032
=================================================================================================
```

系数不显著

图 7-6

可以看到并不是每个虚拟变量都是显著的。“Never-worked”和“Without-pay”这两个类别所对应的虚拟变量并不显著，表示这两个类别与基准类别并无差异，应该从模型中移除相应的虚拟变量。

（1）如程序清单 7-2 里的第 8～10 行代码所示，枚举出定性变量里的所有类别，一共有 9 个不同的类别。类别在数组“1”里的顺序决定了对应虚拟变量的顺序，其中数组的第一个值为基准类别。

（2）如第 12～14 行代码所示，定义矩阵“contrast”。这个矩阵有 9 行，每一行按顺序对应着数组“1”中的一个类别。其中前 3 行是零向量，对应着类别“？”“Never-worked”和“Without-pay”；矩阵有 6 列，表示将生成 6 个新的虚拟变量（一共 9 个类别，需要除去 1 个基准类别和 2 个不显著类别），并使用“ContrastMatrix”对新生成的虚拟变量命名。

（3）如第 15～17 行代码所示，定义模型。模型字符串“formula”将被第三方库 patsy[7]处理，生成对应的统计模型。其中“C(workclass, contrast_mat, levels=l)”表示将定性变量“workclass”，按数组“1”的顺序生成虚拟变量，每个类别对应的虚拟变量值由 contrast_mat 定义。

程序清单 7-2 定性变量

```
1 | from patsy import ContrastMatrix
2 |
3 | def trainModel2(data):
```

[7] patsy 是用于描述统计模型的第三方库。统计模型库 Statsmodels 依赖于此库，所以在安装 Statsmodels 时，已经安装好了 patsy。限于篇幅，关于 patsy 的使用细节，这里就不做展开了，有兴趣的读者可参考其官网。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

```
    """
    加入 workclass 变量，搭建逻辑回归模型，并训练模型
    """
    # 定义 workclass 的类别顺序，数组里的第一个值为基准类别
    l = [' ?', ' Never-worked', ' Without-pay', ' State-gov',
        ' Self-emp-not-inc', ' Private', ' Federal-gov',
        ' Local-gov', ' Self-emp-inc']
    # 定义各个类别对应的虚拟变量
    contrast = np.eye(9, 6, k=-3)
    # 为每个虚拟变量命名
    contrast_mat = ContrastMatrix(contrast, l[3:])
    formula = """label_code ~ C(workclass, contrast_mat, levels=l)
        + C(sex) + education_num + capital_gain
        + capital_loss + hours_per_week"""
    model = sm.Logit.from_formula(formula, data=data)
    re = model.fit()
    return re
```

运行程序清单 7-2，得到的结果如图 7-7 所示。

```
                           Logit Regression Results
==============================================================================
Dep. Variable:             label_code   No. Observations:                26048
Model:                          Logit   Df Residuals:                    26036
Method:                           MLE   Df Model:                           11
Date:                Sun, 04 Jun 2017   Pseudo R-squ.:                  0.2642
Time:                        21:42:02   Log-Likelihood:                -10574.
converged:                       True   LL-Null:                       -14370.
                                        LLR p-value:                     0.000
==================================================================================================
                                                   coef    std err          z      P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------------------------
Intercept                                       -7.5784      0.145    -52.287      0.000      -7.862      -7.294
C(workclass, contrast_mat, levels=l) State-gov   0.3520      0.130      2.699      0.007      0.096       0.608
C(workclass, contrast_mat, levels=l) Self-emp-not-inc  0.3687  0.116  3.165  0.002  0.140  0.597
C(workclass, contrast_mat, levels=l) Private     0.5058      0.101      4.984      0.000      0.307       0.705
C(workclass, contrast_mat, levels=l) Federal-gov 1.0792      0.132      8.196      0.000      0.821       1.337
C(workclass, contrast_mat, levels=l) Local-gov   0.6012      0.118      5.106      0.000      0.370       0.832
C(workclass, contrast_mat, levels=l) Self-emp-inc  1.2921  0.129  10.026  0.000  1.040  1.545
C(sex)[T. Male]                                  1.2097      0.045     27.092      0.000      1.122       1.297
education_num                                    0.3284      0.008     41.737      0.000      0.313       0.344
capital_gain                                     0.0003   1.08e-05     30.770      0.000      0.000       0.000
capital_loss                                     0.0007   3.62e-05     20.617      0.000      0.001       0.001
hours_per_week                                   0.0287      0.002     18.937      0.000      0.026       0.032
==================================================================================================
```

调整后的虚拟变量

图 7-7

7.2.3 从定性变量到定量变量

前面讨论的虚拟变量的方法是比较通用的处理方法。但这种方法有一个很明显的缺点：每个虚拟变量都是 0 或 1，无法提供更多的信息。特别是对于多个有序的定性变量，这会损失掉每个定性变量本身的顺序信息和定性变量间的关联信息。为了解决这个问题，常常根据类别的顺序，将定性变量转换为定量变量。具体的转换方法有很多，但限于篇幅，这里只讨论其中的一种：针对二元分类问题的 Ridit scoring[8]，如图 7-8 所示。

[8] 此方法在保险业中应用很广。具体的细节参考自 Brockett P L, Derrig R A, Golden L L, et al. Fraud Classification Using Principal Component Analysis of RIDITs[J]. Journal of Risk & Insurance, 2002, 69(3):341–371.

仅供非商业用途或交流学习使用

假设有序的定性变量x有t个可能的取值，记为$(1,2,\ldots,t)$。而且对于被预测值y，排在后面的类别，$y=1$发生的可能性越小。也就是说，对于$y=1$这件事，其他变量相同时，类别1的概率最大，类别t的概率最小。用$(p_1,p_2,\ldots,p_t)$分别表示各个类别所占比例，于是类别i的Ridit scoring 为：

$$B_i = \sum_{j<i} p_j - \sum_{j>i} p_j \tag{7-3}$$

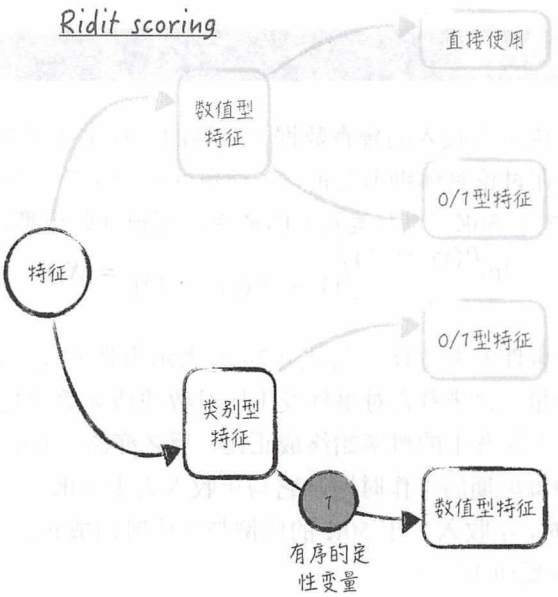


图 7-8

根据公式（7-3），可以将定性变量转换为定量变量。比如对于性别变量，由于社会上存在的性别歧视，在同等条件下，男性工资高于女性工资。而在数据集里男性的比例为 0.67，女性的比例为 0.33。因此$B_{Male} = -0.33, B_{Female} = 0.67$。具体的代码可参考程序清单 7-3。

程序清单 7-3 Ridit scoring

```
def trainModel3(data):
    """
    使用 sex 变量的 Ridit scoring 搭建逻辑回归模型，并训练模型
    """
    l = [" Male", " Female"]
    contrast = [[-0.33], [0.67]]
    contrast_mat = ContrastMatrix(contrast, ["Ridit(sex)"])
    formula = """label_code ~ C(sex, contrast_mat, levels=l) + education_num
        + capital_gain + capital_loss + hours_per_week"""
    model = sm.Logit.from_formula(formula, data=data)
```

仅供非商业用途或交流学习使用

```
    re = model.fit()
    return re
```

如果模型中只有一个有序的定性变量，Ridit scoring 的优势并不明显。当有多个有序变量时，Ridit scoring 的效果明显优于虚拟变量。具体的细节超出了本书的范围，这里就不做展开了。

7.3 定量变量的处理

本章我们依然以美国个人收入的普查数据（数据的具体字段可参考 7.2.2 节）为基础讨论对定量变量的处理。在讨论具体细节之前，先简单回顾一下第 5 章中的逻辑回归模型。用 $P(y_i = 1)$表示年收入大于 50K（5 万美元）的概率，逻辑回归模型可以表示为如下形式。

$$\ln {P(y_i = 1)}/{[1 - P(y_i = 1)]} = \boldsymbol{X_i\beta} \tag{7-4}$$

$P/(1 - P)$被称为事件的发生比，公式（7-4）表示事件的发生比对数为线性模型。如果直接使用一个定量变量，意味着它对事件发生比对数的边际效应是恒定的。当它在模型中的系数大于 0 时，它与事件发生的概率始终成正比，反之亦然。但这个隐含的模型假设并不始终与现实相符，比如每星期的工作时间，它与年收入大于 50K 的交叉报表如图 7-9 所示：当工作时间小于 80 小时，年收入大于 50K 的比例与工作时间成正比；而大于 80 小时之后，年收入大于 50K 的比例反而下降了。

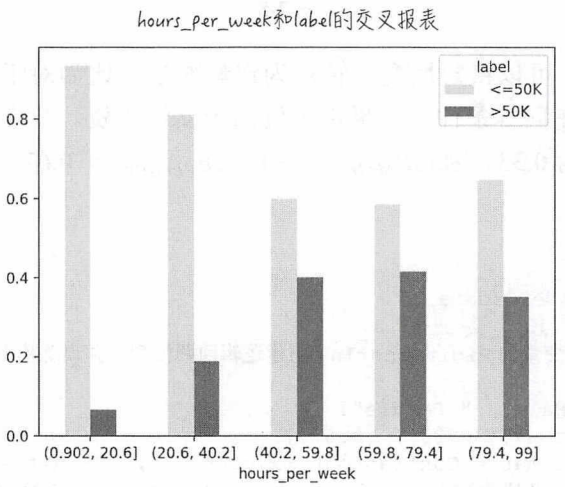


图 7-9

仅供非商业用途或交流学习使用

从数学上看，隐含的边际效应恒定假设来源于两个方面，一是线性模型，二是直接使用定量变量。正如前面章节讨论的，复杂模型经过一层层分解总能得到线性模型。也就是说规避问题的第一个源头是很困难的，几乎不可能。因此为了解决问题，只能从第二个源头入手：将定量变量按取值区间转换为定性变量。然后按照定性变量的第一种处理方法，生成多个虚拟变量替代转换后的定性变量。这是在实践中最常用也最有效的方法，下面将讨论这个方法的具体细节。

7.3.1 定量变量转换为定性变量

为了将定量变量转换为定性变量，将定量变量的取值范围划分为几个互不相交的子区间。每个子区间对应着一个类别，并以此生成新的定性变量。整个过程如图 7-10 所示。这个方法设计的思路是，定量变量对预测值的影响在一定范围内可以近似为恒定的，而这些“恒定”的范围对应着划分的子区间。在实践中，子区间的划分常常依赖于业务场景的专家知识和建模者的个人经验。

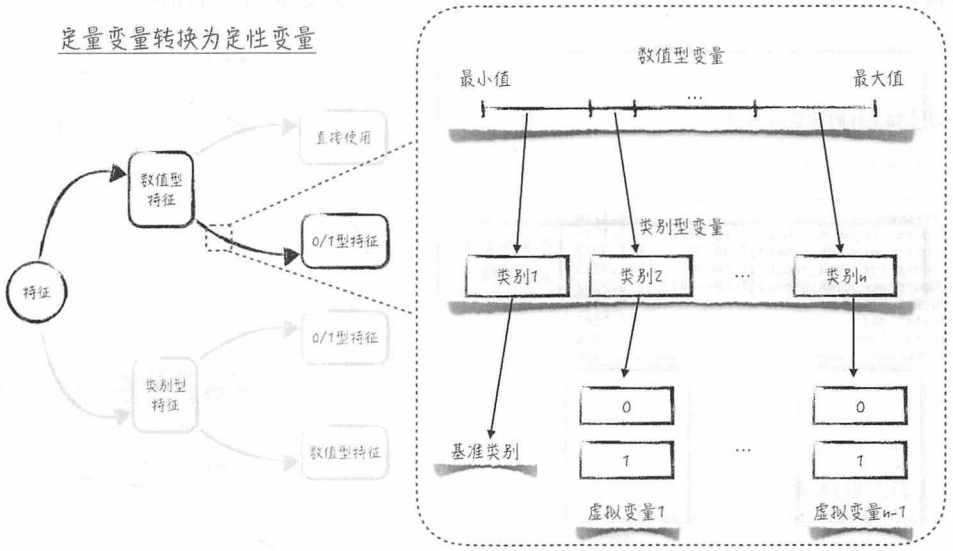


图 7-10

7.3.2 上手实践：代码实现

以每星期的工作时间为例，如程序清单 7-4 所示，将其等分为 5 个子区间，并相应地生成定性变量“hours_per_week_group”，它的取值如第 9～13 行代码所示。

程序清单 7-4 定量变量转换为定性变量

```
1 | >>> category = range(0, 105, 20)
2 | [0, 20, 40, 60, 80, 100]
```

仅供非商业用途或交流学习使用

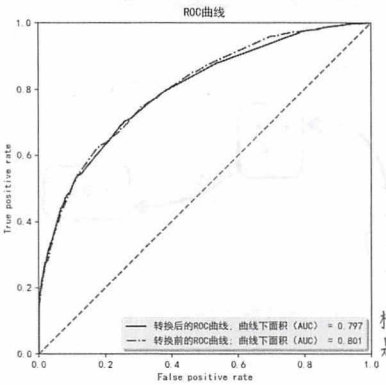
```
>>> labels = ["{0}-{1}".format(category[i], category[i+1]) for i in range(len(category) - 1)]
['0-20', '20-40', '40-60', '60-80', '80-100']
>>> data["hours_per_week_group"] = pd.cut(data["hours_per_week"],
...         category, include_lowest=True, labels=labels)
>>> data[["hours_per_week", "hours_per_week_group"]].head()
   hours_per_week hours_per_week_group
0              40               20-40
1              13                0-20
2              40               20-40
3              40               20-40
4              40               20-40
```

用新变量“hours_per_week_group”替代原有变量，得到的模型结果虽然符合预期，但模型效果反而下降了，如图 7-11 所示。这是因为当使用定量变量时，该变量的任何差别都会影响模型最终的预测结果。但如果将其转换为了定性变量，在同一类别里，这个变量的差别将不会对预测结果起作用。比如使用了新变量后，每星期工作 30 小时和每星期工作 40 小时，年收入大于 50K 的概率是一样的（当然，这是在其他变量相同的情况下）。

每星期工作时间等分为5份

	coef
Intercept	-6.1776
C(hours_per_week_group)[T.20-40]	1.1916
C(hours_per_week_group)[T.40-60]	1.9972
C(hours_per_week_group)[T.60-80]	1.9353
C(hours_per_week_group)[T.80-100]	1.7762
education_num	0.3126
capital_gain	0.0003
capital_loss	0.0008

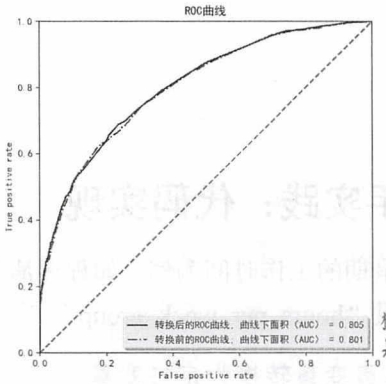
变量系数与图7-9相符



模型效果下降

每星期工作时间等分为10份

	coef
Intercept	-5.8862
C(hours_per_week_group)[T.10-20]	-0.4022
C(hours_per_week_group)[T.20-30]	-0.1443
C(hours_per_week_group)[T.30-40]	1.0024
C(hours_per_week_group)[T.40-50]	1.6946
C(hours_per_week_group)[T.50-60]	1.7782
C(hours_per_week_group)[T.60-70]	1.6794
C(hours_per_week_group)[T.70-80]	1.6088
C(hours_per_week_group)[T.80-90]	1.7718
C(hours_per_week_group)[T.90-100]	1.2554
education_num	0.3116
capital_gain	0.0003
capital_loss	0.0008



模型效果上升

图 7-11

仅供非商业用途或交流学习使用

因此，当子区间划分不是很合理时，会损失掉很多原变量的信息。比如，上面将每星期工作时间等分成 5 份，就显得划分得太“粗”了。如果将其划分得更“细”一点，比如等分成 10 份，模型的预测效果就将上升。但是区间划分得越细，引入的虚拟变量也就越多。相比于之前，模型变得更复杂了。这使得模型训练（得到模型参数的估计值）更加困难，而且很容易引起过度拟合的问题。

那么，有没有一种比较工程化的方法来帮助我们划分区间呢？答案是肯定的。下面将介绍在银行信贷模型中应用很广的分段方法。

7.3.3 基于卡方检验的方法

对定量变量分段的原则是，分段后的生成的类别型变量 A 与被预测量 B 越相关越好。换句话说，A 与 B 越不相互独立越好。

针对分类问题，被预测量 B 也是类别型变量。而对于两个类别型变量，卡方检验（chi-squared test）可以用来度量它们之间的相关性：定义并计算变量 A 与 B 的卡方统计量，这个量越大，两个变量越相关[9]。具体的检测步骤如下。

- 定义两个变量之间的列联表（contingency table），也称为交叉报表。列联表是一个二维列表，行表示一个变量，列表示另一个变量，如图 7-12 所示。

Contingency Table(列联表)

变量A的类别

变量B的类别

	A1	A2	A3	总计
B1	1	1	3	5
B2	1	1	10	12
总计	2	2	13	17

类别A3,B2包含10个数据

图 7-12

- 从列联表中，可以得到各个类别的实际值。在变量 A 与 B 相互独立的假设下，可以计算每个类别应该包含的数据个数，即预测值。比如对于 A_3 和 B_1，我们有$P(A = A_3) = 13/17, P(B = B_1) = 5/17$，因此这个类别的预测值为$5/17 \times 13 \approx 3.82$。针对这一“格”，定义统计量：

[9] 卡方检验从本质上讲是假设检验。零假设为两个变量相互独立，卡方统计量越大，相应的 P-value 就越小。当 P-value 低于某个阈值时，就可以拒绝相互独立的零假设。因此可以近似地理解为：卡方统计量越大，两个变量越相关。

仅供非商业用途或交流学习使用

$$T_{3,1} = (\text{实际值} - \text{预测值})^2 / \text{预测值} = (3.82 - 3)^2 / 3.82 \tag{7-5}$$

- 将公式（7-5）定义的每一“格”统计量相加，得到卡方统计量$T = \sum T_{i,j}$。T服从自由度为$(3-1) \times (2-1) = 2$的卡方分布[10]。

有了上面的理论基础，可以利用贪心算法得到每星期工作时间的“最优”分段，如程序清单 7-5 所示。

（1）将每星期工作时间划分成最优的两段：从最小值开始，依次遍历所有可能的分段，并从中找出对应卡方统计量最大的一个，如第 18～34 行代码所示。其中，第 25～27 行代码生成列联表，第 28 行代码利用第三方库 scipy 计算相应的卡方统计量。

（2）递归地调用解决二分问题的函数 divideData，直至得到的卡方统计量小于某个阈值，如第 4～15 行代码所示。

程序清单 7-5 基于卡方检验的分段方法[11]

```
 1 | import pandas as pd
 2 | import scipy.stats as scs
 3 |
 4 | def doDivide(data, interval):
 5 |     """
 6 |     使用贪心算法，得到“最优”的分段
 7 |     """
 8 |     category = []
 9 |     pValue, chi2, index = divideData(data, interval[0], interval[1])
10 |     if chi2 < 15:
11 |         category.append(interval)
12 |     else:
13 |         category += doDivide(data, [interval[0], index])
14 |         category += doDivide(data, [index, interval[1]])
15 |     return category
16 |
17 | def divideData(data, minValue, maxValue):
18 |     """
19 |     遍历所有可能的分段，返回卡方统计量最高的分段
20 |     """
21 |     maxChi2 = 0
22 |     index = -1
23 |     maxPValue = 0
24 |     for i in range(minValue+1, maxValue):
```

[10] 卡方分布（chi-square distribution）是概率论与统计学中常用的一种概率分布。假设$Z_1, Z_2, \ldots, Z_k$表示k个相互独立的标准正态分布变量（期望为 0，标准差为 1），定义变量X如下：

$$X = \sum_{i=1}^{k} Z_i^2$$

X服从自由度为k的卡方分布。

[11] 完整的实现请参考随书配套的代码/ch07-econometrics/continuous_variable.py。

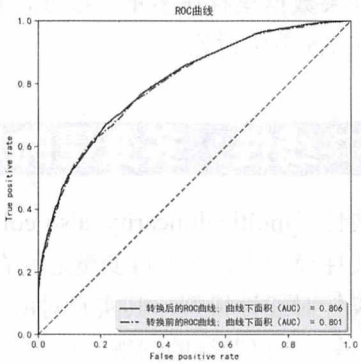
仅供非商业用途或交流学习使用

```
        category = pd.cut(data["hours_per_week"], [minValue, i, maxValue],
            include_lowest=True)
        cross = pd.crosstab(data["label"], category)
        chi2, pValue, _, _ = scs.chi2_contingency(cross)
        if chi2 > maxChi2:
            maxPValue = pValue
            maxChi2 = chi2
            index = i
    return maxPValue, maxChi2, index
```

根据上面的方法，得到的“最优”解将每星期的工作时间分为 5 段，分别为 1～34、34～37、37～41、41～49、49～99。利用这个分段结果重新搭建模型，可以得到更好的结果，如图 7-13 所示。

基于卡方检验，将每星期工作时间分成5段

	coef
Intercept	-6.1007
C(hours_per_week_group)[T.34-37]	0.7750
C(hours_per_week_group)[T.37-41]	1.2699
C(hours_per_week_group)[T.41-49]	1.7780
C(hours_per_week_group)[T.49-99]	1.9936
education_num	0.3118
capital_gain	0.0003
capital_loss	0.0008



模型效果上升

图 7-13

7.4 显著性

当我们使用某些变量搭建模型时，其实隐含了两个假设：第一，这些被使用的变量与被预测量是真正相关的，即变量能在某种程度上决定被预测值；第二，如果这里面包含了与被预测量无关的变量，那么它们对应的模型参数应该为 0。这两个假设在数学上是完全没有问题的，但在实际生产中，模型参数是由求解最优化问题的算法估算出来的，比如第 6 章中讨论的随机梯度下降法。这些算法并不关心变量与被预测量是否真的相关，任何变量都能得到相应的参数估计值。而且由于随机扰动的影响，即使毫不相关的变量放进模型里，其参数的估计值也几乎不可能等于 0，这就是所谓的模型幻觉。对应的解决方法就是借助线性回归或逻辑回归模型，考查模型参数的显著性，将不显著的变量排除出模型。这些问题的细节在第 4 章和第 5 章里已经详细讨论过，此处就不再重复了。

值得注意的是，对于模型参数a，它的显著性依赖于两方面，一是参数的估计值，用$\hat{a}$表

书籍是人类进步的阶梯

示；二是参数估计值的标准差估计值，用$\widehat{se(\hat{a})}$表示。当$|\hat{a}| < k * \widehat{se(\hat{a})}$时（$k$的取值与显著性水平相关，比如$k = 1.96$对应5%的显著性水平），就认为这个参数不显著，相应的变量将被移出模型，反之，则保留在模型中。估计值$\hat{a}$和$\widehat{se(\hat{a})}$都是根据训练数据估算出来的。因此，它们都具有一定的随机性，而且还受模型其他因素的影响。在考查变量的显著性时，如果忽略了这些因素，就很容易得到错误的结论，比如将重要的相关变量错误地排除出模型。为了保证显著性判断的准确性，相应地需要保证两个条件。一是估计值$\hat{a}$的期望等于真实值a，即$E(\hat{a}) = a$；二是$\hat{a}$的标准差估计值$\widehat{se(\hat{a})}$接近于理论值[12]。

第一个条件这在学术上被称为不偏的估计值（consistent estimator）。但在实际生产中，如果模型的变量存在内生性，这个条件将不再成立。具体的细节将在7.6节中讨论。对于第二个条件，如果数据里存在多重共线性，估计值$\widehat{se(\hat{a})}$就会被急剧扩大，不再满足要求。具体的细节将在7.5节中讨论。

7.5 多重共线性：多变量的烦恼

多重共线性（multicollinearity, also collinearity）源自对线性回归模型的深入研究。它是指在多变量线性模型中，由于自变量之间存在高度相关关系而使模型参数估计不准确。多重共线性不仅仅针对线性模型，其实它对所有模型的参数估计都有影响（具体原因将在7.5.1节里讨论），是一个很普遍的问题。但由于线性模型的分析工具更多、更全面，在搭建其他模型时，我们也常常借助线性模型来检测和处理自变量间存在的多重共线性问题。因此，本节的讨论将集中在线性回归模型上。

7.5.1 多重共线性效应

我们通过一个人造的数据集来看看多重共线性的效应。数据集中包含一个被预测量y和3个自变量x_1, x_2, x_3。被预测量和变量之间的关系如公式（7-6）所示，其中，ε是随机扰动项。当然在搭建模型时，我们并不知道这个公式。

$$y = 0.7x_1 - 1.1x_2 + 0.3x_3 + \varepsilon \tag{7-6}$$

其中，变量x_1和x_2完全不相关，相关系数[13]等于0；变量x_1和x_3强相关，相关系数等于

[12] 假设在线性模型里，随机扰动项的方差为σ^2，而参数a对应的变量为x，则$\widehat{se(\hat{a})}$的理论值为$\sigma^2/\mathrm{Var}(x)$。

[13] 这里的相关系数是指皮尔逊积矩相关系数（Pearson product-moment correlation coefficient）。它常被用于度量两个变量X和Y之间的相关（线性相关）。取值为−1～1，1表示完全正相关，即X和Y在一条直线上，且直线朝上；−1表示完全负相关，即X和Y在一条直线上，且直线朝下。具体的定义公式如下：

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X\sigma_Y} = \frac{E[(X - E[X])(Y - E[Y])]}{\sigma_X\sigma_Y}$$

其中，$\mathrm{Cov}(X,Y)$表示X和Y的协方差，σ_X, σ_Y分别表示X和Y的标准差。

书籍是人类进步的阶梯

0.958，具体如图 7-14 所示。

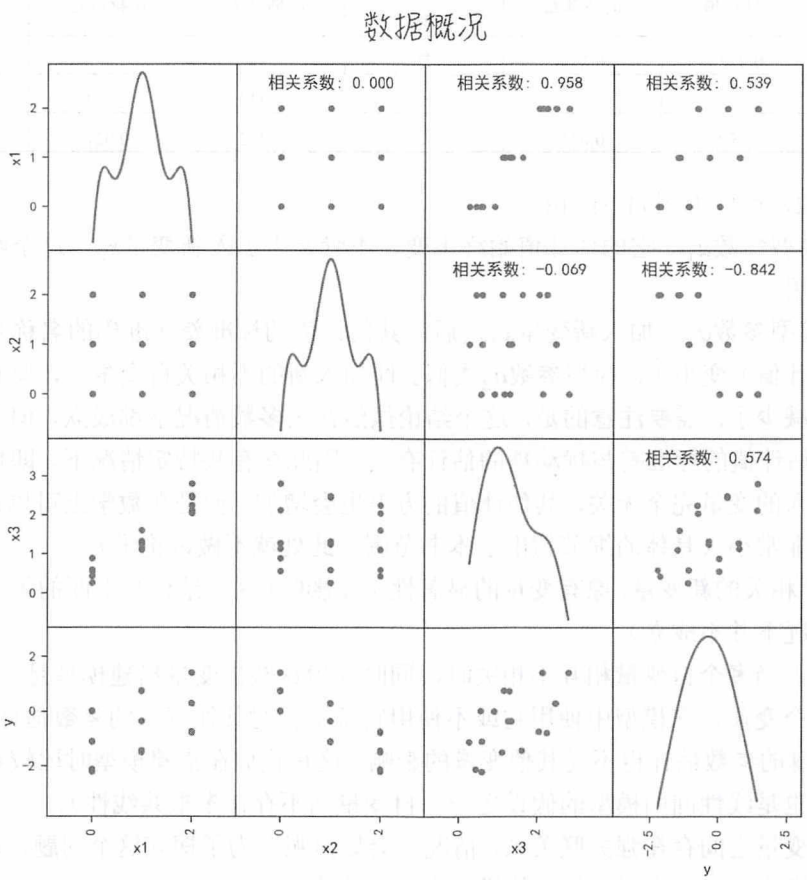


图 7-14

首先使用不相关的x_1和x_2分别搭建 3 个不同的线性回归模型，具体的模型形式如公式（7-7）所示。

$$y = a_1x_1 + b + \varepsilon$$
$$y = a_2x_2 + b + \varepsilon$$
$$y = a_1x_1 + a_2x_2 + b + \varepsilon \tag{7-7}$$

这 3 个模型参数的估计结果列举在表 7-2[14]中。

[14] 具体的代码实现请参考随书配套的代码/ch07-econometrics/multicollinearity.py。

书籍是人类进步的阶梯

表 7-2

模型	a_1估计值	a_1标准差	a_1是否显著（5%）	a_2估计值	a_2标准差	a_2是否显著（5%）
只有x_1	0.972	0.279	是	—	—	—
只有x_2	—	—	—	−1.113	0.244	是
x_1和x_2	0.972	0.021	是	−1.113	0.021	是

根据表 7-2，可以得到如下结论。

- 对于模型参数a_1，它的估计值始终不变，不管是否加入新变量x_2，这个结论对模型参数a_2同样适用。
- 对于模型参数a_1，加入新变量x_2之后，其估计值的标准差（准确的名称是参数估计值的标准差估计值）变小了；模型参数a_2类似。即加入新的不相关自变量后，原有自变量估计值的标准差减少了。需要注意的是，这个结论虽然在大多数情况下都成立，但并非没有例外，模型参数估计值的标准差与扰动项的估计有关。因此在有些特定情况下，即使原有的某个变量与新加入的变量完全无关，其估计值的方差也会增加。但是在数学上可以证明，方差增加的幅度会非常小（具体的细节超出了本书范围，此处就不做讨论了）。
- 加入不相关的新变量，原有变量的显著性不受影响（这一结论与上面的第二点类似，在某些极端情况下并不成立）。

由此可见，当多个自变量相互不相关时，同时使用这些自变量搭建模型是“安全”的。对于其中的某个变量，在模型中使用它或不使用它都不会对其他变量的参数估计产生影响。反过来，它自身的参数估计也不受其他变量的影响。这是我们在搭建模型时比较希望看到的情况（这其实也是线性回归模型的假设之一：自变量间不存在多重共线性）。

但如果自变量之间存在强关联关系，情况又会如何呢？为了探究这个问题，我们使用强相关的x_1和x_3搭建类似的模型，具体的模型形式如公式（7-8）所示。

$$y = a_1x_1 + b + \varepsilon$$
$$y = a_3x_3 + b + \varepsilon$$
$$y = a_1x_1 + a_3x_3 + b + \varepsilon \tag{7-8}$$

模型的结果如表 7-3 所示。

表 7-3

模型	a_1估计值	a_1标准差	a_1是否显著（5%）	a_3估计值	a_3标准差	a_3是否显著（5%）
只有x_1	0.972	0.279	是	—	—	—
只有x_3	—	—	—	1.090	0.282	是
x_1和x_3	−0.168	0.959	否	1.260	1.017	否

书籍是人类进步的阶梯

表中的结果与之前的结论有很大的不同，具体如下。

- 模型参数a_1的估计值不仅跟变量x_1有关，还与模型中是否有变量x_3有关。加入新变量x_3后，a_1的估计值有比较正确的 0.972 变成了偏差比较大的−0.168。
- 在加入新变量x_3后，模型参数a_1的标准差估计值也大幅增加了，从原来的 0.273 变成了 0.959。换句话说，加入新的强相关自变量后，原有自变量的标准差变大了（同样地，这个结论也只是在绝大多数情况下成立。）。
- 原本模型参数a_1是显著的，但加入自变量x_3后，a_1变得不显著了。这个结论在很大程度上，是上面第二点的直接推论。

到此为止，我们对多重共线性的讨论集中在它对模型参数的影响。但如果从另外一个角度来看，它对模型的预测效果是否有影响呢？答案是比较模糊的。一方面，多重共线性会使模型参数的估计值不准确，这显然会影响模型预测的准确性；但另一方面，就线性回归模型本身而言，在某种程度上，多重共线性并不影响模型的预测效果。

比如公式（7-8）的模型效果如表 7-4 所示。加入强相关变量后，模型决定系数是上升的。它表示线性模型对已知数据（即训练模型时使用的训练数据）的预测效果几乎不受多重共线性的影响。这与模型参数的估计值形成鲜明对比。需要注意的是，上述结论只针对线性模型以及已知数据。

表 7-4

模型	决定系数R^2
只有x_1	0.431
只有x_3	0.483
x_1和x_3	0.484

总结一下，多重共线性对模型有如下 4 种效应。

- 参数的估计值变得不准确[15]。
- 参数估计值的标准差变大。
- 参数显著性检验变得不准确，容易将重要的自变量误判为不显著。从更大层面上来讲，多重共线性使得针对模型参数的假设检验变得不准确。
- 对于已知数据，模型的预测效果几乎不受影响。

我们可以从更直观的角度来理解第一种和第二种效应：变量的参数代表了变量对被预测量的影响。当模型中的自变量都不相关或者不太相关时，自变量的变化并没有什么联动效应，

[15] 这个结论在学术上是不太严谨的。因为在多重共线性下，参数的估计值其实是无偏的。也就是说估计值的期望依然等于真实值。但由于估计值的标准差变大，所以在绝大多数情况下，具体的估计值会远离其真实值。从这个层面上来讲，我们可以说多重共线性使参数估计值变得不准确。

书籍是人类进步的阶梯

可以简单地理解为，每一对数据之间，只有一个自变量发生变化，因此可以比较容易地分辨出被预测量的变化来源于哪里。但当模型中存在两个或多个强相关的自变量时，由于这些自变量总是同时变动，所以很难分清被预测量的变化具体来自哪个变量。比如图 7-15，x和z不同时变化时，就很容易找出y和这两个变量的关系。但若x_1和z_1与之相反，它们同时变化，y_1和它们的关系就难以确定。

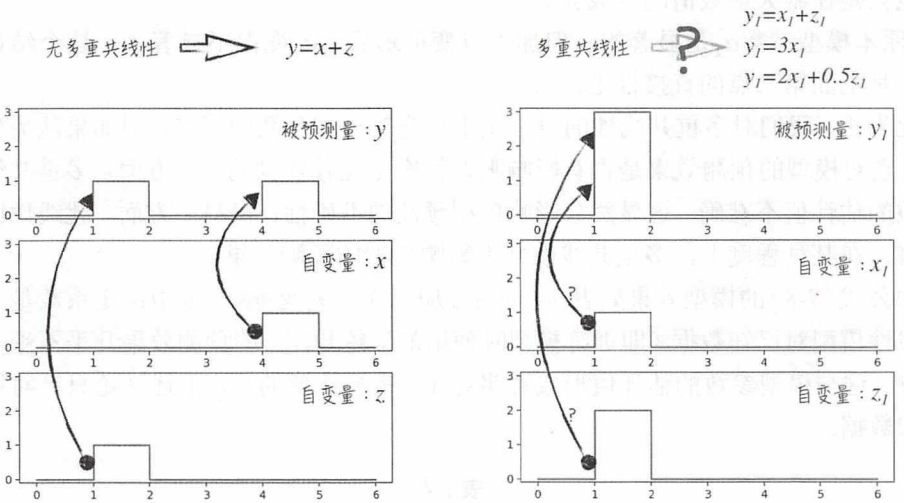


图 7-15

由此可见，前面两种效应对所有模型都是适用的，同时也是很严重的问题。而后面的两种效应主要针对线性模型或广义线性模型，如逻辑回归（因为其他模型并不提供类似的分析工具）。

7.5.2 检测多重共线性

本节将讨论如何检测变量间可能存在的共线性问题。如果考查的是两个变量，通常借助相关性指标来做判断；如果考查的是多个变量，则相应的共线性检测常常基于线性模型的假设检验以及方差膨胀因子（variance inflation factor）。

针对两个变量，我们根据它们之间的相关性指标来判断是否存在共线性问题，具体的方法可以分为 3 种。

（1）考查的两个变量全是定量变量，则依赖的相关性指标为相关系数以及相应的 P-value[16]。在实践中，通常选用皮尔逊积矩相关系数（Pearson correlation）或者斯皮尔曼等

[16] 基于假设检验的 P-value：在真实的相关系数等于 0 的假设下，计算得到的相关系数所对应的 P-value。

书籍是人类进步的阶梯

级相关系数（Spearman correlation）[17]。这两种相关系数的取值范围都为$[-1,1]$，其中等于 0 或者接近于 0 表示两个变量几乎相互独立；靠近 1 表示两个变量正相关，且相关性很强；靠近−1 与靠近 1 类似，只是表示负相关。因此，若变量间的相关系数明显不等于 0（可以参考基于相关系数的 P-value 来判断相关系数是否明显不等于 0，具体的判断标准与假设检验一样），则可以认定数据中存在共线性问题，反之则不然。比如在上一节的数据中，x_1和x_3的相关系数为 0.958，很接近于 1（从另一个角度，相关系数的 P-value 等于 4.09e−10。这表示相关系数显著不等于 0），它们之间的确存在共线性问题。相应的实现请参考程序清单 7-6 中的第 1～10 行代码。

程序清单 7-6 相关系数

```
 1 | >>> import scipy.stats.stats as scss
 2 | >>> # 计算皮尔逊积矩相关系数。
 3 | >>> # 返回值有两个，第一个是相关系数，第二个是两个变量不相关这个假设的 P-value
 4 | >>> # data 为 7.5.1 节中的数据，包含变量“x1” “x2” “x3” “y”
 5 | >>> scss.pearsonr(data["x1"], data["x2"])
 6 | (0.0, 1.0)
 7 | >>> scss.pearsonr(data["x1"], data["x3"])
 8 | (0.95821805640469349, 4.0954797021801555e-10)
 9 | >>> # 计算斯皮尔曼等级相关系数。返回值也有两个，第一个是相关系数，第二个是 P-value
10 | >>> scss.spearmanr(data["x1"], data["x2"])
11 | SpearmanrResult(correlation=0.0, pvalue=1.0)
```

（2）考查的两个变量全是定性变量，则依赖的指标为卡方检验中的卡方统计量和相应的 P-value。具体的计算步骤和实现请参考 7.3.3 节。

（3）考查的变量里既有定量变量 A，又有定性变量 B，则依赖的指标为 one-way ANOVA 分析中的η^2。one-way ANOVA 是一种假设检验技术，它主要用于检测多组数据的均值是否显著不同。如图 7-16 所示，图中有两组数据“d1”和“d2”，它们同为期望等于 5 的正态分布，但实际计算得到的均值并不相同。经过 one-way ANOVA 分析，“d1”和“d2”两组数据的均值并没有显著不同。

---

[17] 斯皮尔曼等级相关系数（Spearman's rank correlation coefficient）是统计中常用的相关性指标。它利用单调方程评价两个统计变量的相关性。其取值为−1～1，如果数据中没有重复值， 并且当两个变量完全单调相关时，斯皮尔曼相关系数则为 1 或 −1。具体的计算公式如下：

$$\gamma_s = \frac{\mathrm{Cov}(rg_X, rg_Y)}{\sigma_{rg_X}\sigma_{rg_Y}} = \rho_{rg_X, rg_Y}$$

其中，rg_X, rg_Y分别表示变量X和Y的排序序号。

书籍是人类进步的阶梯

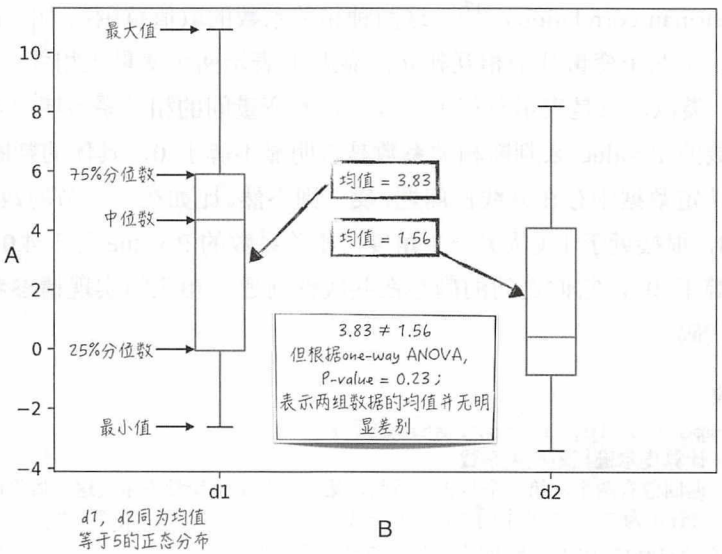


图 7-16

回到多重共线性检测，我们将定量变量 A 当成被预测量、定性变量 B 当成自变量搭建线性回归模型，并以此为基础进行 one-way ANOVA 分析。技术指标η^2可以近似地理解为变量 A 的变化中有多少部分可以由变量 B 解释[18]（在某种程度上，可以将η^2视为 A 与 B 的“相关系数”）。因此，当η^2接近于 1 时，可以认为这两个变量间存在共线性问题，反之则不然。具体的实现可参考程序清单 7-7。

程序清单 7-7 one-way ANOVA[19]

```
import statsmodels.api as sm

def oneWayANOVA(data):
    """
    计算定量变量 A 与定性变量 B 之间的 eta squared

    参数：
    ----
```

[18] 假设定性变量 B 有k个类别，于是将定量变量 A 按 B 的取值分为了k组。可以证明 A 的方差（记为SS_{total}）可以分解为两部分：不同组之间的方差（即每组平均值的方差，记为$SS_{between}$）和组内部的方差之和（即将每组的方差相加起来，记为SS_{within}）。于是定义η^2如下：

$$\eta^2 = \frac{SS_{between}}{SS_{total}}$$

η^2的取值介于 0 与 1 之间。当它等于 1 时，表示在一个组内，A 恒等于某个值，即 A 的取值完全由 B 的取值决定。

[19] 完整的代码实现请参考随书配套的代码/ch07-econometrics/one_way_ANOVA.py。

书籍是人类进步的阶梯


```
    data : DataFrame, 包含变量 A 和变量 B
    """
    re = sm.OLS.from_formula("A ~ B", data=data).fit()
    aovTable = sm.stats.anova_lm(re, typ=2)
    # 打印 ANOVA 分析结果
    print aovTable
    # 计算 eta sqaured
    etaSquared = aovTable["sum_sq"][0] / (aovTable["sum_sq"][0] + aovTable["sum_sq"][1])
    print "Eta squared 等于: %.3f" % etaSquared
```

针对多个变量，常用的方法有两种：联合假设检验和方差膨胀因子。需要注意的是，这两种方法可用于检测两个或多个变量之间的共线性问题。但是上面所讨论的相关性指标只适用于两个变量。为了书写简洁，下面仍以两个变量作为例子讨论。

1. 基于线性回归模型的假设检验

先回到 7.5.1 节中的例子，正如表 7-3 所总结的，由于x_1和x_3之间存在强相关关系，所以当它们同时出现在模型中时，它们都分别被误判为不显著。但如果检验x_1和x_3同时不显著（即联合假设检验），得到结论却是否定的，具体的检验结果如图 7-17 所示。这反过来给了我们检测多重共线性的方法：某几个变量分别不显著，而它们又联合显著。这就表示这几个变量间存在多重共线性问题。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.415
Method:                 Least Squares   F-statistic:                     7.040
Date:                Sun, 11 Jun 2017   Prob (F-statistic):            0.00698
Time:                        23:27:18   Log-Likelihood:                -22.989
No. Observations:                  18   AIC:                             51.98
Df Residuals:                      15   BIC:                             54.65
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -1.5437      0.622     -2.480      0.025      -2.870      -0.217
x1            -0.1677      0.959     -0.175      0.864      -2.213       1.877
x3             1.2604      1.017      1.240      0.234      -0.907       3.428
==============================================================================
Omnibus:                        3.209   Durbin-Watson:                   2.726
Prob(Omnibus):                  0.201   Jarque-Bera (JB):                1.239
Skew:                           0.079   Prob(JB):                        0.538
Kurtosis:                       1.724   Cond. No.                         14.5
==============================================================================

检测假设x1和x3同时不显著:
<F test: F=array([[ 7.04011618]]), p=0.00697750891904, df_denom=15, df_num=2>
```

1 x1, x3分别不显著

2 拒绝x1和x3同时不显著的联合假设

图 7-17

书籍是人类进步的阶梯

书籍是人类进步的阶梯

2. 方差膨胀因子

这个技术指标能衡量模型参数估计值的方差被膨胀了多少倍。为了说明这一点，我们假设如下的线性回归模型。

$$y = \beta_0 + \beta_1 x_1 + \ldots + \beta_k x_k + \varepsilon \tag{7-9}$$

其中，ε的方差是σ^2。以参数β_1为例（其他参数类似），其估计值的估计方差记为$\widehat{Var}(\hat{\beta}_1)$，数学上可以证明：

$$\widehat{\mathrm{Var}}(\hat{\beta}_1) = \frac{\sigma^2}{\mathrm{Var}(x_1)} \frac{1}{1 - R_1^2} \tag{7-10}$$

公式（7-10）中的R_1^2是公式（7-11）这个线性回归模型的决定系数。

$$x_1 = \alpha_0 + \alpha_2 x_2 + \ldots + \alpha_k x_k + e \tag{7-11}$$

正如第 4 章中讨论的，线性模型的决定系数是衡量模型解释力的指标。决定系数R_1^2越接近 1，线性模型的效果越好。换句话说，变量x_1几乎能被其他变量完全解释，这表示数据中的多重共线性问题越严重，而且x_1是引起共线性问题的变量之一。反之，若R_1^2接近于 0，则表示在模型里加入x_1不会引起共线性问题。

于是针对x_i定义方差膨胀因子VIF_i，如公式（7-12）所示。

$$VIF_i = {}^{1}/_{(1 - R_i^2)} \tag{7-12}$$

这个指标的具体实现可参考程序清单 7-8。值得注意的是，在计算 VIF 时，不要忘记在数据里加入常数项（常数项对应着线性回归模型里的截距项，比如公式（7-11）中的α_0），如第 7 行代码所示。

程序清单 7-8 方差膨胀因子

```
 1 | >>> from statsmodels.stats.outliers_influence import
variance_inflation_factor
 2 | >>> import pandas as pd
 3 | >>>
 4 | >>> # data 为 7.5.1 节中的数据，包含变量 "x1"、"x2"、"x3"、"y"
 5 | >>> X = data[["x1", "x2", "x3"]]
 6 | >>> # 在数据里加入常数项（如果无）
 7 | >>> X = sm.add_constant(X)
 8 | >>>
 9 | >>> vif = pd.DataFrame()
10 | >>> vif["VIF Factor"] = [variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
11 | >>> vif["features"] = X.columns
12 | >>> print vif
13 |    VIF Factor features
14 | 0   11.049590    const
15 | 1   12.907932       x1
16 | 2    1.061102       x2
17 | 3   12.969034       x3
```

通常当方差膨胀因子大于 5 时，就可以认定对应的变量有比较明显的共线性问题。如程

书籍是人类进步的阶梯

书籍是人类进步的阶梯

序清单 7-8 中第 13～17 行代码所示，x_1和x_3的方差膨胀因子都大于 5，它们之间也的确存在比较严重的共线性问题。

以上就是检测多重共线性问题的常用方法。如果检测到数据中存在这样的问题，那应该如何解决它呢？这是下一节将探讨的内容。

7.5.3 解决方法

为了讨论方便，本节将沿用 7.5.1 节中的数据集。在探讨具体的解决方法之前，先将引起多重共线性的原因分类。可以将其分为两类：源于数据的共线性和结构化共线性。之前章节一直讨论的都是源于数据的共线性，这也是最常见的一类。而结构化共线性是指，由特征提取中数学变换引起的共线性。比如基于变量x_3，生成新的变量x_3^2，x_3和x_3^2之间很容易引发共线性问题。针对这两类问题，有各自相应的解决方法。

1．常用的解决方法

源于数据的多重共线性是一个比较难解决的问题，并没有一个特别工程化或者说特别通用的方案。需要具体场景，具体分析。这里将简单介绍 5 种常用的解决方法。

（1）增加数据量

这是最有效，也是最无效的解决方法。

一方面，多重共线性对模型的危害在于，它将增加模型参数估计值的方差，使参数估计值变得不稳定。而增加数据量能从整体上降低参数估计值的方差，从而克服共线性带来的危害。如图 7-18 所示，虽然x_1和x_3之间存在很严重的共线性问题，但随着数据量的增加，a_1的估计值越来越接近真实值 0.7，其标准差的估计值也逐步减小。因此从模型层面考虑，这是最有效的解决方法。这也是大数据的魅力所在，通过数据的增加解决原本在数学上很难处理的问题。

另一方面，数据量受很多实际因素的限制，比如收集数据的成本等。而且很多时候，几乎不可能拿到更多的数据。因此增加数据量这个话题已经超出了搭建模型的范畴，更像是一个商业问题或者说经济问题。站在这个层面上讲，这也是最无效的解决方法。

（2）去掉某些引起共线性的不重要变量

比如在我们使用的数据集中有 3 个变量x_1, x_2, x_3，其中x_1和x_3强相关。可以把相对不那么重要的x_3排除出模型，这样模型就不再有共线性的问题了。这种方法的思路是，既然变量之间存在强相关关系，那么，其中的某个变量的信息几乎能被其他变量所替代，比如x_3几乎等于x_1。所以将其中一部分不太重要的变量排除出模型，并不会带来太大的信息损失。这的确是一种解决办法，但其实是一种很危险的做法。其原因有两个：一是很难判断变量的重要性，很可能误将最重要的变量排除出模型；二是这样做总归会损失一些变量信息，而且损失的信息量难以量化。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

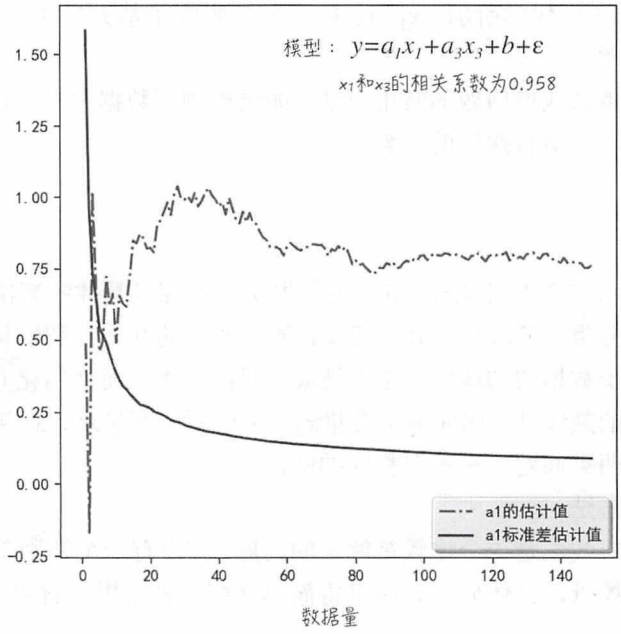


图 7-18

（3）将数据降维

我们用新生成的数量相对较少的变量替代原本的变量，比如根据x_1和x_3生成新变量x_4，然后在模型里用x_4同时替代x_1和x_3。对数据降维要求新生成的变量能尽可能地减少信息损失，且不存在共线性问题。满足这个要求的方法有很多，最为常见的是主成分分析（PCA），具体的算法细节将在第 10 章中讨论。

（4）加入惩罚项

如果只针对线性回归模型，可以通过加入惩罚项来消除共线性带来的影响。具体地，在线性回归模型中加入 L2 惩罚项（即 Ridge 回归）能比较有效地解决共线性问题，相关的代码请参考第 4.3.2 节。

这里我们想从直观上来解释一下为什么加入惩罚项是有效的？以线性模型$y = a_1x_1 + a_3x_3 + \varepsilon$为例。从空间上来讲，这个模型就是在一个三维空间里（相应的坐标分别为y, x_1, x_3）找出一个平面，使得这个平面离数据点的距离之和达到最小。不妨叫找到的平面为“最佳平面”。由于x_1和x_3之间的强相关关系，数据点几乎呈一条直线。这使得“最佳平面”缺乏额外的点来固定它，很容易晃动。反映到模型上就表现为模型不稳定，参数估计值的方差很大。但加入惩罚项之后，就好像有一条绳子连接着原点和“最佳平面”。有了原点的加入，“最佳平面”多了一个强有力的固定点，晃动的幅度会大幅度减小。换句话说，模型参数更稳定，

书籍是人类进步的阶梯

书籍是人类进步的阶梯

消除了共线性的影响。

（5）鸵鸟政策（ostrich policy）[20]

简单来讲，所谓的鸵鸟政策就是忽略共线性问题。这也许是一个不是办法的办法，但在某些情况下这也是合理的。比如对于已知数据，模型的预测效果几乎不受共线性的影响。因此在这种情况下，完全可以忽略共线性问题的存在。

2．结构化共线性

为了表述简洁，我们以x_3和x_3^2为例讨论这种情况的处理。

（1）将变量的中心重置为 0（centering data）

首先来看看结构化共线性的来源，如图 7-19a 所示。在远离原点的地方，x_3^2和x_3几乎呈一条直线分布，也就是说它们之间的相关系数接近于 1，这就是问题的来源。相应的解决方法其实图中也已经给出了答案，就是让变量的值尽量靠近原点，如图 7-19b 所示。

于是，我们用$x_3 - \overline{x_3}$替代x_3；$(x_3 - \overline{x_3})^2$替代x_3^2。其中，$\overline{x_3}$表示变量x_3的平均值。注意到，变量$x_3 - \overline{x_3}$的平均值等于 0。因此，该方法被称为将变量中心重置为 0。在线性模型中，由于截距项的存在，模型对自变量的线性变换（对变量做常数加减乘除运算）是稳定的，具体的论证请参考第 4.1.1 节。因此将变量的中心重置为 0 对模型效果并没有影响。当然更好一点的做法，是将变量归一化，即用$(x_3 - \overline{x_3})/\mathrm{std}(x_3)$替代$x_3$；$(x_3 - \overline{x_3})^2/\mathrm{Var}(x_3)$替代$x_3^2$。稍加注意可以得到，变量$(x_3 - \overline{x_3})/\mathrm{std}(x_3)$的方差等于 1，这是我们称该方法为变量归一化的原因。经过归一化后，变量会更加集中于原点附近。而且各个变量的变化幅度大致相同，这在工程上将有利于模型参数的估算[21]。因此，在建模实践中，不管是否存在多重共线性问题，通常都会对变量进行归一化处理。

分别使用x_3, x_3^2和$x_3 - \overline{x_3}, (x_3 - \overline{x_3})^2$对被预测量$y$进行建模，具体的模型形式如公式（7-13）所示：

$$y = ax_3 + bx_3^2 + c + \varepsilon$$
$$y = a(x_3 - \overline{x_3}) + b(x_3 - \overline{x_3})^2 + c + \varepsilon \tag{7-13}$$

模型训练后得到的结果见表 7-5。可以看到，对变量做了处理之后，模型参数的估计值几乎不变，但参数估计值的方差都变小了。与此同时，两个模型的决定系数相同，也就是说模型效果没有变化。这些迹象表明模型的共线性问题消失了。

---

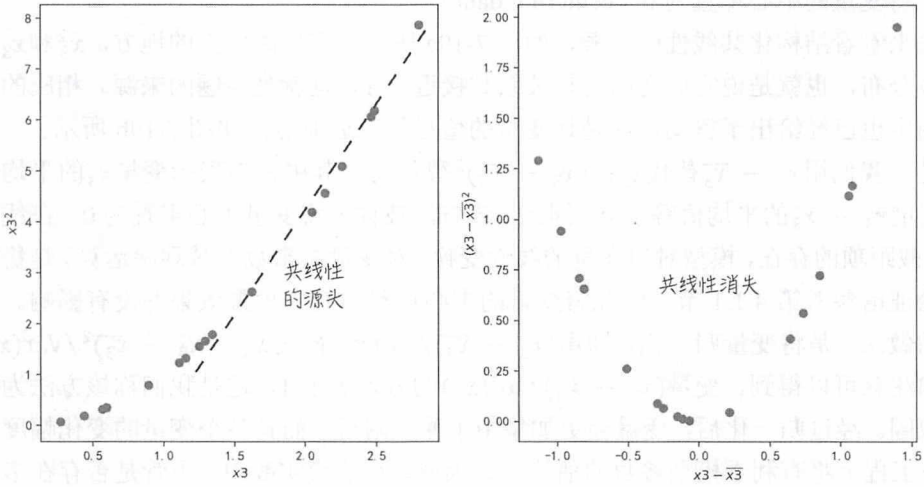
[20] 根据维基百科的记载，当鸵鸟在遇到危险的时候，习惯于闭上眼睛，把头埋进土里。当鸵鸟看不到危险的时候，就相信危险不存在。但是其身体仍然暴露在外面，并且在应对危险的时候更加脆弱。因此，鸵鸟政策指面对危险的时候，采取放任的态度。

[21] 在第 6 章中，我们讨论了模型参数的估算方法——随机梯度下降法，这个算法中有一个很重要的参数叫学习速率（learning rate）。这个参数对所有变量都是相同的，所以当各个变量的变化幅度大致相同时，算法的收敛速度最快，收敛效果最好。

书籍是人类进步的阶梯

表 7-5

模型	a估计值	a标准差	a是否显著（5%）	b估计值	b标准差	b是否显著（5%）	决定系数
x_3, x_3^2	1.191	1.367	否	−0.033	0.442	否	0.483
$x_3-\overline{x_3}$, $(x_3-\overline{x_3})^2$	1.097	0.305	是	−0.033	0.442	否	0.483



（a） （b）

图 7-19

（2）上面针对源于数据的多重共线性问题，一共列举了 5 种常用的方案。这些方案对结构化共线性也是适用的，这里就不再赘述了。

总结一下，针对多重共线性，常用的解决方法有上述的 6 种。其中除了变量归一化（或者将变量中心重置为 0）比较通用以外，其他方法都有比较明确的适用范围。在实际生产中，我们需要根据具体场景，选取适合的一个或多个方法来消除共线性的影响。

7.5.4 虚拟变量陷阱

本节我们将焦点重新放到定性变量上，讨论一种由它带来的多重共线性。正如 7.2.1 节中讨论的，在模型中使用多维定性变量时，需要将其转换为多个虚拟变量。如果处理不当，这么多个虚拟变量之间就很容易发生共线性问题，这种现象在学术上被称为“虚拟变量陷阱”（dummy variable trap）。

书籍是人类进步的阶梯

针对虚拟变量陷阱，之前已经讨论过一种很浅显的情形。这里先简单回顾一下，对于一个n维的定性变量，将其转换为n个虚拟变量。由于这n个虚拟变量之和一定等于 1，所以同时使用它们一定会带来很严重的共线性问题。事实上，这些变量之间存在绝对意义上的线性关系，所以会导致模型参数根本无法估计。为了解决这个问题，我们会从定性变量的n个类别里选取一个作为基准类别，并基于此将n维定性变量转换为$n-1$个虚拟变量。

这个方法看似解决了问题。但如果基准类别选择不合理，生成的虚拟变量之间依然会存在严重的多重共线性。为了说明这一点，我们来看看下面这个例子。

（1）首先生成一个 4 维的定性变量“categoricalData”，4 个类别分别为“a”“b”“c”和“d”，其中类别“a”的占比最小，只有 5 个，如程序清单 7-9 里的第 5 行代码所示。然后将这个变量转换为 4 个虚拟变量，分别为“Var_a”“Var_b”“Var_c”和“Var_d”，如第 6 行代码所示。

（2）以类别“a”作为基准类别（默认情况下，开源算法库，如 Statsmodels，会使用字母序的第一个类别作为基准类别），即保留虚拟变量“Var_b”“Var_c”和“Var_d”，如第 26～33 行代码所示。然后计算这 3 个变量的方差膨胀因子（具体的计算函数如第 15～23 行代码所示），如第 34～39 行代码所示。可以得到“Var_b”和“Var_c”的方差膨胀因子分别为 10.2 和 9.6，均超过 5。这表示它们之间的共线性问题很严重。

（3）为什么会发生上面这种情况呢？答案是基准类别“a”的占比太少，而类别“b”和类别“c”的占比大致相同。这个结论的严格数学证明比较复杂，这里就不展开了。但从直观上可以这样理解：当“categoricalData”不等于类别“a”时，“Var_b”“Var_c”和“Var_d”3 个变量之和等于 1， 否则它们之和不等于 1。所以，在绝大多数情况下，上面的 3 个变量在同一平面上，即它们之间存在强相关关系，这会导致它们之间存在严重的多重共线性问题。

（4）数学上可以证明，选择占比最大的类别作为基准类别就可以解决上面的问题。比如选择类别“b”（占比最大）作为基准类别，即保留变量“Var_a”“Var_c”和“Var_d”。这些变量的方差膨胀因子都在 1 附近，这表示同时使用这 3 个变量不会引起共线性问题。具体的实现请参考第 41～54 行代码。

程序清单 7-9 虚拟变量陷阱

```
1 | >>> import pandas as pd
2 | >>> import statsmodels.api as sm
3 | >>> from statsmodels.stats.outliers_influence import variance_inflation_factor
4 | >>> # 生成定性变量，并将其转换为虚拟变量
5 | >>> categoricalData = ["a"] * 5 + ["b"] * 100 + ["c"] * 70 + ["d"] * 20
6 | >>> data = pd.get_dummies(categoricalData, prefix="Var")
7 | >>> data.head()
8 |    Var_a  Var_b  Var_c  Var_d
9 | 0    1.0    0.0    0.0    0.0
```

书籍是人类进步的阶梯

```
10 |  1    1.0    0.0    0.0    0.0
11 |  2    1.0    0.0    0.0    0.0
12 |  3    1.0    0.0    0.0    0.0
13 |  4    1.0    0.0    0.0    0.0
14 |  >>>
15 |  >>> def computeVIF(data):
16 |  ...     """
17 |  ...     计算变量的方差膨胀因子
18 |  ...     """
19 |  ...     data = sm.add_constant(data)
20 |  ...     vif = pd.DataFrame()
21 |  ...     vif["VIF Factor"] = [variance_inflation_factor(data.values, i) for
i in range(data.shape[1])]
22 |  ...     vif["features"] = data.columns
23 |  ...     return vif
24 |  ...
25 |  >>> # 选取"a"作为基准类别
26 |  >>> df = data[["Var_b", "Var_c", "Var_d"]]
27 |  >>> df.head()
28 |     Var_b  Var_c  Var_d
29 |  0    0.0    0.0    0.0
30 |  1    0.0    0.0    0.0
31 |  2    0.0    0.0    0.0
32 |  3    0.0    0.0    0.0
33 |  4    0.0    0.0    0.0
34 |  >>> computeVIF(df)
35 |     VIF Factor features
36 |  0   39.000000    const
37 |  1   10.230769    Var_b
38 |  2    9.615385    Var_c
39 |  3    4.487179    Var_d
40 |  >>> # 选取"b"作为基准类别
41 |  >>> df1 = data[["Var_a", "Var_c", "Var_d"]]
42 |  >>> df1.head()
43 |     Var_a  Var_c  Var_d
44 |  0    1.0    0.0    0.0
45 |  1    1.0    0.0    0.0
46 |  2    1.0    0.0    0.0
47 |  3    1.0    0.0    0.0
48 |  4    1.0    0.0    0.0
49 |  >>> computeVIF(df1)
50 |     VIF Factor features
51 |  0    1.950000    const
52 |  1    1.023077    Var_a
53 |  2    1.089744    Var_c
54 |  3    1.076923    Var_d
```

将上面例子的经验总结一下，可以得到避免虚拟变量陷阱的方法：对于一个n维定性变量，选择其中占比最大的类别作为基准类别，并生成$n-1$个虚拟变量分别代表剩余的类别。

书籍是人类进步的阶梯

7.6 内生性：变化来自何处

与多重共线性类似，内生性（endogeneity）同样也源自线性模型。从数学的角度上来看，它表示由于模型中的一个或多个变量与随机扰动项相关，从而导致模型参数估计不准确（参数估计值的期望不等于真实值）。从直观角度上理解，这其实是由于被预测量与自变量互为因果，从而导致了内生性。在某种意义上，内生性其实有点类似“先有鸡，还是先有蛋”这种哲学式的思考。因此，它又是充满争议，难有标准答案的一个问题。

举一个现实中的例子，重点中小学和学生的学业表现。在中国，家长们都希望自己的孩子能进入所谓的重点中小学，接受更好的教育。这一举动甚至催生了一个很热门的房地产概念——学区房。那么问题来了，重点学校是否真的提供了更好的教育，从而帮助学生达到更好的学业表现呢？从表面上看，重点学校的升学率和平均成绩都明显优于其他学校。但这些差距是由重点学校本身带来的吗？还是因为好学校本身有一个逆向选择的过程，将优秀的学生聚集到这里，所以才有了这样的差距？或者更近一步，进入重点中小学和学业表现优异都是某些遗漏变量导致的结果，比如家庭条件等，两者之间其实并没有因果关系？而内生性就是用于帮助我们回答这些问题的。

由于国内相关的研究较少，我们来看看欧美的情况。大量的研究结果[22]表明在其他条件相同的情况下，比如父母的教育背景、家庭收入等，就读学校是否是名校对学生的学业表现没有任何显著的影响。倒是上面提到的这两个因素对学生的学业和是否进入名校都有明显的正向影响。将这些结果翻译成模型语言，以是否进入重点中学作为自变量x，学业表现作为被预测量y，如果对这两个变量进行建模，会发现x和y之间有很显著的正相关关系。但如果针对变量x做内生性的特殊处理，就会得到截然不同结论，即x和y之间没有显著的相关关系。

回到内生性本身。在实际生产中，当需要借助模型来解释现象，理解各个变量对结果的影响时，是否存在内生性问题就显得特别重要。因为没能正确处理它，会导致得到的结论有偏差。因此它一直是计量经济学的研究热点和前沿领域。有很多相关问题至今也没有一个让各方都信服的解决方法。正因为如此，本节针对内生性的讨论将集中于相对简单的线性回归模型，并简要介绍针对逻辑回归模型的传统方法[23]。

---

[22] 相关的文献实在太多，这里仅列举一个例子：Cullen J B, Jacob B A, Levitt S. The Effect of School Choice on Participants: Evidence from Randomized Lotteries[J]. Econometrica, 2006, 74(5):1191-1230.

这篇文章通过对美国芝加哥地区的研究，发现在相同条件下，进入名校的学生并没有在学业上表现得更好，反而因为竞争压力表现得更差。

[23] 本节所涉及的具体实现请参考随书配套的代码/ch07-econometrics/endogeneity.py。

首先来看看内生性的来源。

7.6.1 来源

内生性问题常见的来源有3个[24]：遗漏变量（omitted variable）、度量误差（measurement error）、自变量和被预测量的同时性（simultaneity）。

下面以线性回归模型为例，分别介绍它们。

（1）首先讨论遗漏变量。假设有如下的线性关系：

$$y_i = ax_i + bz_i + c + \varepsilon_i \tag{7-14}$$

其中，x_i和随机扰动项ε_i相互独立，但x_i和z_i并不相互独立。而且z_i是被遗漏的变量，也就是说在搭建模型时，只有变量y_i, x_i。所以我们搭建的模型如公式（7-15）所示：

$$y_i = ax_i + c + u_i \tag{7-15}$$

公式（7-15）中，u_i是随机扰动项。由于$u_i = bz_i + \varepsilon_i$，所以$x_i$可以通过$z_i$影响$u_i$的取值，也就是说$x_i$和$u_i$之间并不相互独立。这就是有遗漏变量带来的内生性。

（2）再来看看由度量误差引起的内生性。假设有自变量x_i和被预测量y_i，它们之间的关系如下所示：

$$y_i = ax_i + b + \varepsilon_i \tag{7-16}$$

但由于测量时的误差v_i，x_i的观测值x_i^*并不等于真实值，而是等于真实值加上观测误差，即$x_i^* = x_i + v_i$。由于在搭建模型时，得到的变量就是x_i^*，因此实际的模型如公式（7-17）所示：

$$\begin{aligned} y_i &= ax_i^* + b + \varphi_i \\ \varphi_i &= \varepsilon_i - av_i \end{aligned} \tag{7-17}$$

可以看到，$x_i^* = x_i + v_i$和$\varphi_i = \varepsilon_i - av_i$都有随机变量$v_i$。所以自变量$x_i^*$和随机扰动项$\varphi_i$并不相互独立，会带来内生性问题。具体地，度量误差会使得参数a被低估。

（3）最后介绍自变量和被预测量的同时性，以及由此带来的内生性。考虑如下两个线性模型：

$$\begin{aligned} y_i &= a_1x_i + b_1z_i + \varepsilon_i \\ z_i &= a_2x_i + b_2y_i + \varphi_i \end{aligned} \tag{7-18}$$

当$b_1b_2 \neq 1$时，可以求出z_i的表达式：

$$z_i = \frac{a_2 + b_2\ a_1}{1 - b_1\ b_2}x_i + \frac{1}{1 - b_1\ b_2}\varphi_i + \frac{b_2}{1 - b_1\ b_2}\varepsilon_i \tag{7-19}$$

所以z_i和扰动项ε_i并不相互独立，会在一个模型里引入内生性问题。对于第二个模型，用类似的推导方法，可以得到相同的结论。

上面介绍的3种来源中，遗漏变量和度量误差几乎在每个建模场景中都能被遇到，因此

[24] 参考自维基百科。

内生性问题其实广泛存在，是影响模型准确性的“罪魁祸首”。

7.6.2 内生性效应

本节将讨论内生性对模型的影响。为了更直观地理解，我们根据上面提到的度量误差，人为地制造出有内生性问题的数据集。具体地，数据集里有被预测量y、自变量$real_x_1$和它的观测值x_1以及另一个自变量x_2。它们之间的关系如公式（7-20）所示。其中，ε为随机扰动项，φ为度量误差，而且$real_x_1, x_2, \varepsilon, \varphi$这 4 个变量相互独立。

$$y = 10real_x_1 + 20x_2 + 30 + \varepsilon$$
$$x_1 = real_x_1 + \varphi \tag{7-20}$$

由于变量$real_x_1$是不可知的，所以建模时面对的数据为y, x_1, x_2。这三者之间的关系如公式（7-21）所示：

$$y = 10x_1 + 20x_2 + 30 + error \tag{7-21}$$

如果用图形表示$x_1, x_2, error$之间的关系，就可以得到如图 7-20 所示的结果。注意到x_1和$error$之间存在强相关关系（相关系数为–0.896），这会引起模型的内生性问题。

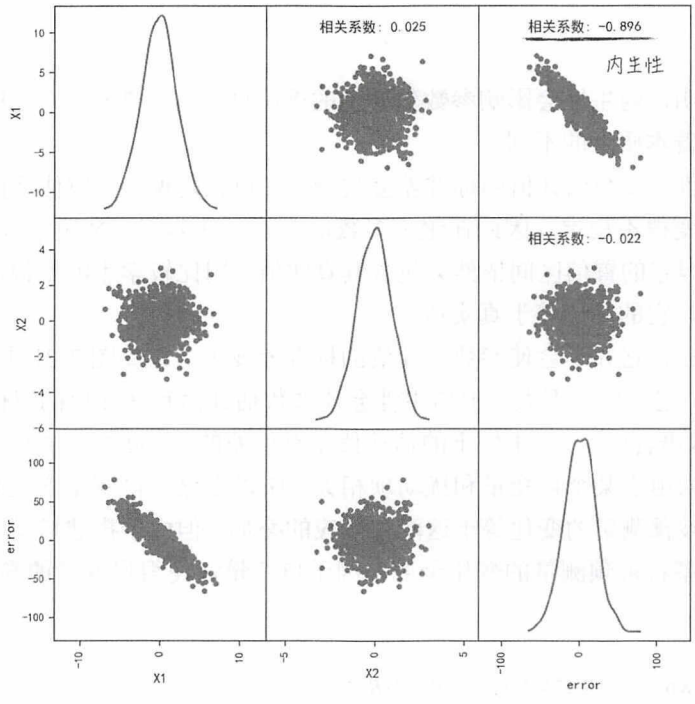


图 7-20

使用变量x_1和x_2对y搭建线性模型，即$y = ax_1 + bx_2 + c + error$。模型的结果如图 7-21 所示。由于内生性，参数a的估计值为 1.796，远离其真实值 10。而且估计值的 95%置信区间为[1.543, 2.049]，也离真实的 10 很远。另一方面，由于x_2与$error$相互独立，所以参数b, c的估计值不受内生性的影响，靠近各自的真实值。

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      Y   R-squared:                       0.825
Model:                            OLS   Adj. R-squared:                  0.824
Method:                 Least Squares   F-statistic:                     2343.
Date:                Sat, 17 Jun 2017   Prob (F-statistic):               0.00
Time:                        09:40:48   Log-Likelihood:                -3617.1
No. Observations:                1000   AIC:                             7240.
Df Residuals:                     997   BIC:                             7255.
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     29.9646      0.286    104.876      0.000      29.404      30.525
X2            20.0024      0.300     66.654      0.000      19.414      20.591
X1             1.7958      0.129     13.934      0.000       1.543       2.049
==============================================================================
```

靠近真实值 ②

① 远离真实值

③ 置信区间也出现偏差

图 7-21

上面的例子说明，内生性会影响参数估计值的准确性，这与 7.5 节中讨论的多重共线性类似，但这两种有着本质上的不同。

由于多重共线性，参数估计值的标准差会被增大。也就是说，参数估计值的变动幅度会增大，估计值本身变得不稳定。因此在绝大多数情况下（并非所有情况下），参数的估计值会远离其真实值，但它的置信区间依然会包含其真实值。用比较学术的话说，多重共线性下的估计值是不偏的，它的期望等于真实值。

对于内生性问题，它并不会使参数估计值的标准差变大。比如图 7-21 中，参数a的标准差估计值为 0.129，这个值并不大。但内生性会使参数估计值和估计值的置信区间同时远离真实值。翻译成学术语言就是，内生性下的估计值是有偏差的，它的期望并不等于真实值[25]。直观上，可以理解为由于某个自变量和扰动项相关，所以当这个自变量变化时，扰动项也相应地跟着变化，而被预测量的变化等于这两种效应的叠加。但由于搭建模型时，只能观测到自变量。因此，模型将被预测量的变化全部归因于自变量，使得自变量的参数被错误估计。

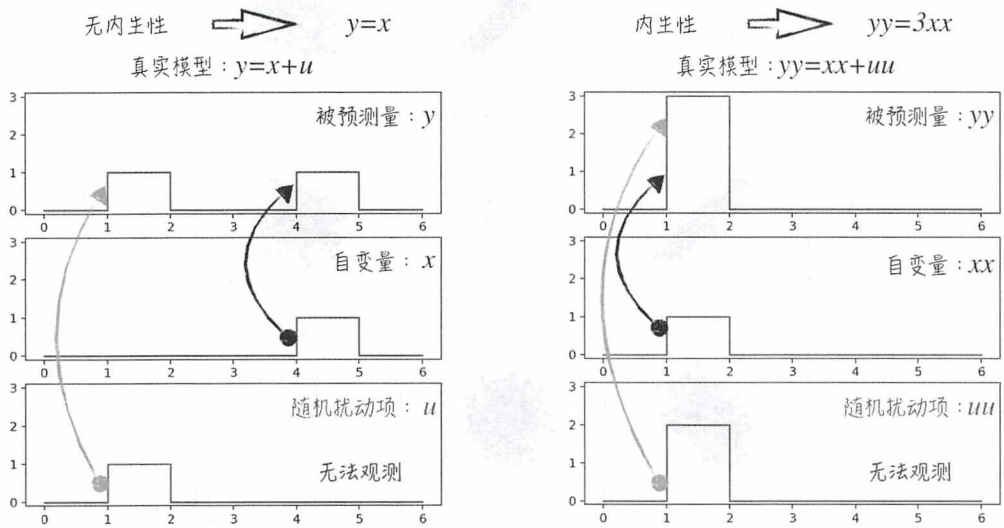
---

[25] 假设模型为$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$，根据推导可以得到参数$\boldsymbol{\beta}$的估计值如下：

$$\widehat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} = \boldsymbol{\beta} + (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\varepsilon}$$

由于$\boldsymbol{X}$与$\boldsymbol{\varepsilon}$相关，即$E(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\varepsilon}) \neq 0$。所以$E(\widehat{\boldsymbol{\beta}}) = \boldsymbol{\beta} + E[(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{\varepsilon}] \neq \boldsymbol{\beta}$。也就是说，参数估计值$\widehat{\boldsymbol{\beta}}$是有偏差的。

比如图 7-22b，xx和uu正相关，当xx增加 1 时，则uu跟着增加 2，所以被预测量yy增加 3。由于无法观测到随机扰动项uu，模型被错误的估计为$yy = 3xx$。



（a）　　（b）

图 7-22

将上面的分析总结一下，多重共线性和内生性都将影响模型的效果，但这两种的“危害”方式并不一样。从某种意义上来说，多重共线性使得模型变得不稳定，而内生性使得模型变得不准确。另外需要注意的是，虽然这两种效应都在线性模型中，但它们对所有模型都是适用的。也就是说不管模型的具体形式如何，如果所用数据中存在多重共线性或内生性，则模型的效果和准确性都会大打折扣。

7.6.3 工具变量

讨论完内生性对模型的影响，我们接下来讨论相应的解决方法——工具变量（instrumental variable）。所谓工具变量就是满足如下两个要求的变量：

- 与引起内生性的自变量存在相关性；
- 与随机扰动项不相关。

找到满足要求的工具变量是十分困难的，就像烹饪一样，需要想象力和灵感。这里不讨论如何寻找工具变量，而是讨论如果找到工具变量后，应该如何使用它解决模型的内生性问题。

针对上面的例子，我们找到了工具变量iv_1，它与x_1相关，但与随机扰动项$error$不相关，如图 7-23 所示。

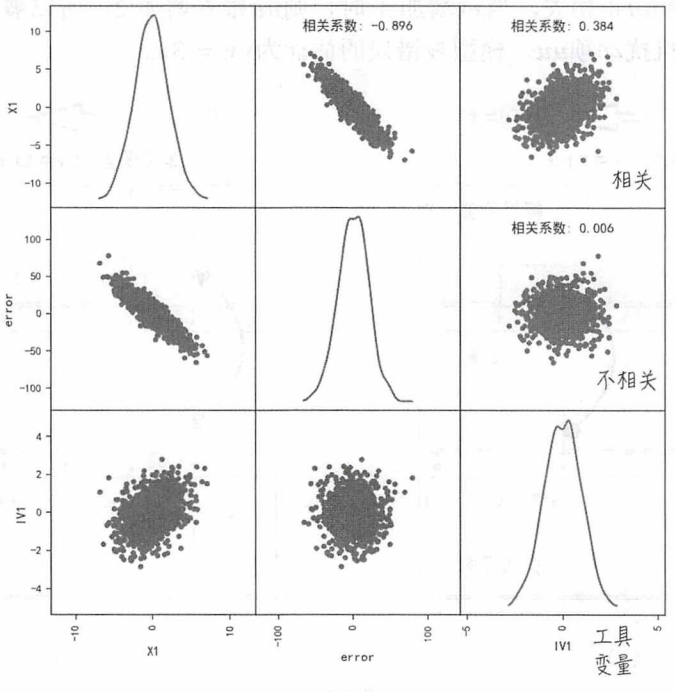


图 7-23

有了工具变量iv_1之后，使用它和变量x_2对x_1搭建线性模型，如公式（7-22）所示：

$$x_1 = aiv_1 + bx_2 + e \tag{7-22}$$

由于iv_1, x_2与$error$无关，而x_1又与$error$相关，那么根据公式（7-22），x_1与$error$的相关性是由随机扰动项e完成的，即$\mathrm{Cov}(x_1, error) = \mathrm{Cov}(e, error)$。那么在原模型（公式（7-21））中加入$x_1$的估计值（或扰动项$e$的估计值），就可以解决模型里的内生性问题[26]。具体的步骤如程序清单 7-10 所示。

（1）如第 6、7 行代码所示，使用变量iv_1和x_2对x_1建模，并求得随机扰动项e的估计值“X1_resid”。

（2）将（1）中得到的“X1_resid”加入原模型，消除内生性问题带来的影响，得到无偏的参数估计值。具体的实现如第 9～11 行代码所示。

---

[26] 假设模型为$\boldsymbol{Y} = \boldsymbol{X\beta} + \boldsymbol{\varepsilon}$、工具变量$\boldsymbol{Z}$以及工具变量与$\boldsymbol{X}$之间的模型$\boldsymbol{X} = \boldsymbol{Z\gamma} + \boldsymbol{e}$。根据线性模型的参数估计公式，我们有$\hat{\boldsymbol{\gamma}} = (\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z})^{-1}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{X}$，并得到$\boldsymbol{X}$的估计值$\hat{\boldsymbol{X}} = \boldsymbol{Z}(\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z})^{-1}\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{X}$。为了书写方便，记$\boldsymbol{P_Z} = \boldsymbol{Z}(\boldsymbol{Z}^{\mathrm{T}}\boldsymbol{Z})^{-1}\boldsymbol{Z}^{\mathrm{T}}$。

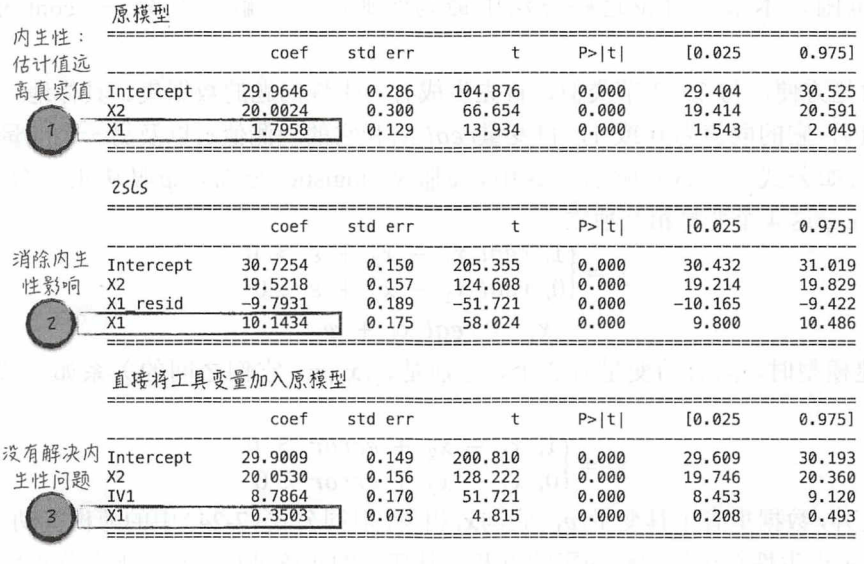
用$\hat{\boldsymbol{X}}$替代$\boldsymbol{X}$代入原模型，得到无偏的参数估计值$\boldsymbol{\beta}_{2SLS} = (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{Y} = \boldsymbol{\beta} + (\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{\varepsilon}$。由于$\boldsymbol{Z}$与$\boldsymbol{\varepsilon}$不相关，所以$E[(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{X})^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{P_Z}\boldsymbol{\varepsilon}] = 0$，即$E[\boldsymbol{\beta}_{2SLS}] = \boldsymbol{\beta}$。

以上的证明过程参考自维基百科。

程序清单 7-10 工具变量

```
def IVRegression(data):
    """
    使用工具变量估计模型参数
    """
    # 第 1 步回归
    re = sm.OLS.from_formula("X1 ~ X2 + IV1", data=data).fit()
    data["X1_resid"] = re.resid
    # 第 2 步回归
    re1 = sm.OLS.from_formula("Y ~ X1 + X2 + X1_resid", data=data).fit()
    print "使用工具变量"
    print re1.summary()
```

上面的这个方法被称为两阶段最小二乘法（Two Stage Least Squares, 2SLS）[27]。模型的结果如图 7-24 所示，使用 2SLS 方法后，内生性对模型的影响被消除了，3 个参数的估计值分别为 10.14、19.52 和 30.72。它们都离各自的真实值很近（真实的参数值分别为 10、20 和 30，请参考公式（7-21））。但值得注意的是，如果直接将工具变量加入原模型，得到x_1对应的参数估计值为 0.35，离真实值 10 更远。这表示直接使用工具变量并不能解决模型的内生性问题。



	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	29.9646	0.286	104.876	0.000	29.404	30.525
X2	20.0024	0.300	66.654	0.000	19.414	20.591
X1	1.7958	0.129	13.934	0.000	1.543	2.049

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	30.7254	0.150	205.355	0.000	30.432	31.019
X2	19.5218	0.157	124.608	0.000	19.214	19.829
X1_resid	-9.7931	0.189	-51.721	0.000	-10.165	-9.422
X1	10.1434	0.175	58.024	0.000	9.800	10.486

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	29.9009	0.149	200.810	0.000	29.609	30.193
X2	20.0530	0.156	128.222	0.000	19.746	20.360
IV1	8.7864	0.170	51.721	0.000	8.453	9.120
X1	0.3503	0.073	4.815	0.000	0.208	0.493

图 7-24

[27] 对于两阶段最小二乘法，可以直接使用第三方库 Statsmodels 提供的函数 IV2SLS。具体的实现请参考随书配套的代码。

上面的讨论集中于如何解决内生性问题。但在面对实际问题时，应该如何识别模型中是否存在内生性问题呢？答案其实隐藏在 2SLS 的计算过程中。如上面所示，2SLS 包含两步回归计算。在第 2 步回归计算中，我们加入了一个新变量，即第 1 步回归的残差预测值，比如“X1_resid”。

- 如果这个变量在第 2 步模型里是显著的，那么模型中存在内生性问题。比如图 7-24 中变量“X1_resid”是显著的，这与原模型存在内生性问题相符。
- 如果该变量不显著，则说明数据里不存在内生性问题。

除此之外，还可以使用 Durbin-Wu-Hausman 检验（Durbin-Wu-Hausman test， Hausman 检验）来检测内生性。Hausman 检验其实是一种假设检验。它的零假设为模型中不存在内生性问题。所以，当检验得到的 P-value 很小时（比如小于 0.01），就可以认定模型中存在内生性问题。

7.6.4 逻辑回归的内生性

逻辑回归的内生性问题是计量经济学里的一个难题。针对它，学术界提出了很多的解决方法，但每种方法都有它的局限性，没有一个通用的解决方案。当然这部分的讨论已经超出了本书的范围，本节只讨论这些方法中最为经典的一个解决方案——control function approach。

为了讨论方便，与 7.6.2 节类似，首先生成有内生性问题的数据集。具体地，数据集里有被预测值y，它的取值为 0 或 1，自变量$real_x_1$和它的观测值x_1以及另一个变量x_2，它们之间的关系如公式（7-23）所示。其中，ε服从 logistic 分布，φ服从正态分布，而且$real_x_1, x_2, \varepsilon, \varphi$这 4 个变量相互独立。

$$y = \begin{cases} 1, & real_x_1 - x_2 + \varepsilon > 0 \\ 0, & real_x_1 - x_2 + \varepsilon \leqslant 0 \end{cases}$$
$$x_1 = real_x_1 + \varphi \tag{7-23}$$

在搭建模型时，面对的变量有 3 个，分别是x_1, x_2, y，它们之间的关系如公式（7-24）所示。

$$y = \begin{cases} 1, & x_1 - x_2 + error > 0 \\ 0, & x_1 - x_2 + error \leqslant 0 \end{cases} \tag{7-24}$$

除此之外，数据里有工具变量iv_1。它与x_1相关，但同公式（7-24）中的随机扰动项$error$无关。为了展示内生性对逻辑回归的影响以及工具变量的正确使用方法，下面分别用 3 种不同的方式搭建模型。

（1）使用变量x_1, x_2对被预测量y搭建模型，具体的模型形式如下：

$$\ln {P(y = 1)}/{P(y = 0)} = ax_1 + bx_2 + c \tag{7-25}$$

仅供非商业用途或交流学习使用

（2）在模型（7-25）的基础上，加入工具变量iv_1。

$$\ln {P(y=1)}/{P(y=0)} = ax_1 + bx_2 + div_1 + c \tag{7-26}$$

（3）Control function approach。这个方法跟线性回归中的 2SLS 类似，整个步骤分为两步，第 1 步使用工具变量iv_1和变量x_2对x_1搭建线性回归模型。

$$x_1 = miv_1 + nx_2 + l + e \tag{7-27}$$

根据第 1 步的回归结果，将公式(7-27)中的随机扰动项e估计值x_1_resid加入模型(7-25)中，得到第 2 步的逻辑回归模型。

$$\ln {P(y=1)}/{P(y=0)} = ax_1 + bx_2 + fx_1_resid + c \tag{7-28}$$

上面 3 种模型的结果如图 7-25 所示。可以看到内生性也会影响参数估计值的准确性，比如公式（7-25）中a的估计值为 0.488，置信区间为[0.378, 0.597]，它们都离真实值 1 很远；直接加入工具变量也无法解决模型的内生性，比如公式（7-26）中a的估计值为 0.244，置信区间为[0.110, 0.378]，它们更加远离真实值 1；但 control function approach 能有效地消除内生性对模型的影响，比如公式（7-28）中a的估计值为 0.879 离真实值 1 很近，而且它的置信区间为[0.703, 1.055]，包含真实值 1。

原模型

内生性：估计值远离真实值

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0133	0.070	-0.189	0.850	-0.151	0.124
X2	-0.8787	0.084	-10.428	0.000	-1.044	-0.714
X1	0.4877	0.056	8.716	0.000	0.378	0.597

1

直接将工具变量加入原模型

没有解决内生性问题

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-0.0253	0.072	-0.354	0.724	-0.166	0.115
X2	-0.9167	0.087	-10.508	0.000	-1.088	-0.746
IV1	0.5946	0.101	5.872	0.000	0.396	0.793
X1	0.2438	0.068	3.570	0.000	0.110	0.378

2

Control function approach

消除内生性影响

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	0.0014	0.072	0.019	0.985	-0.139	0.142
X2	-0.9287	0.088	-10.608	0.000	-1.100	-0.757
X1_resid	-0.6357	0.108	-5.872	0.000	-0.848	-0.423
X1	0.8794	0.090	9.791	0.000	0.703	1.055

3

图 7-25

仅供非商业用途或交流学习使用

最后讨论逻辑回归的内生性检测。与线性回归模型的检测方法类似，在 control function approach 的第 2 步逻辑回归中，第 1 步线性回归的残差预测值被当作新变量加入模型，比如上面的“X1_resid”。如果这个变量在第 2 步模型中是显著的，则原模型中存在内生性问题，反之则不存在。比如在图 7-25 中，变量“X1_resid”是显著的，这与事实相符。

7.6.5 模型的联结

在 7.6.3 节和 7.6.4 节中，我们讨论了针对内生性问题的解决方法。现在换个角度，着重分析一下它们的模型架构。如图 7-26 所示，不管是针对线性回归模型的 2SLS 还是针对逻辑回归的 control function approach，它们都可以分为两层。第一层的模型并不直接预测结果，而是产生中间结果，这些中间结果将作为新变量同原有变量一道被第二层模型使用。

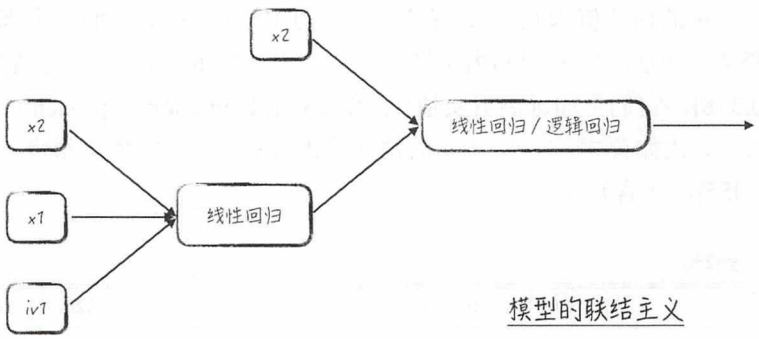


图 7-26

这其实就是所谓的模型联结主义（connectionism），即简单模型的互相连接网络。这个互相连接的模型网络里，线性回归和逻辑回归就像数学里的加减乘除运算一样，是网络里面的原子模型。本书之后的章节将讨论的模型，比如决策数、非监督学习算法等，也都是这种原子模型。但在实际生产中，我们很少会只用一种原子模型来解决问题，而是将多个模型按类似图 7-26 的方式组装起来，形成一个类似网络的“组装”模型。

在本书最后几章将讨论的神经网络或深度学习就是将联结主义发挥到极致的例子。虽然神经网络作为一个整体是非常复杂、让人难以理解的模型，但它里面的原子模型却是非常简单的。比如传统的神经网络其实就是逻辑回归的不断连接嵌套，而深度学习中最常用的原子模型是修正线性模型（rectifier 或 ReLU）。

通常将模型连接起来形成网络后，模型的预测能力会上升（原子模型全是线性回归除外，因为线性模型连接之后依然是线性模型）。但是作为代价，模型的可解释性会下降很多：在一个“组装”模型里，很难量化地分析出某个变量对最终结果的影响如何；“调试”模型也

仅供非商业用途或交流学习使用

会比较困难，比如筛选变量等。因此，原子模型还是“组装”模型，这是一个两难的选择，需要根据不同的问题类型和建模目标，灵活地选择合适的方法。

联结主义所涉及的 3 个方面：更多的原子模型、模型之间常见的连接方式以及神经网络将在后面的章节中进行讨论。

7.7 本章小结

本章并没有介绍具体的模型，而是借鉴计量经济学的思路，讨论了在搭建模型实践中常常遇到的 3 类问题。

首先是特征处理。模型中所用到的变量按能否直接运算可以分为两类。如果一个变量能直接运算，则为定量变量。对于这类变量，可以直接在模型里使用它。但这样的使用方法隐含了变量的边际效应恒定这样一个假设。当需要去除这个隐藏的限制时，则可以将定量变量通过分段的方式转换为定性变量使用。如果一个变量不能直接运算，则为定性变量。为了能在模型里使用这类变量，通常将其转换为多个虚拟变量使用。但对于有序的定性变量，这样处理会损失掉变量本身的顺序信息。特别是当模型中有多个有序定性变量时，损失的信息量就更多了。为了尽可能地保留有序定性变量的信息，可以通过 Ridit score 将定性变量转换为定量变量使用。

其次是某个变量对其他变量以及整体模型的影响。总体而言，变量越多，模型可使用的信息也就越多，其预测效果也会越好。但在有的情况下，变量数目众多并不是一种“祝福”，而更像是一种“诅咒”。变量越多，越容易诱发多重共线性问题，而后者会使得整个模型变得不稳定，影响它的预测能力和解释能力。

最后是随机扰动项对变量的影响。当数据里存在内生性问题时，也就是说某个或某几个变量与随机扰动项相关时，模型的参数估计会受到很大的影响，模型的准确性也无从谈起。内生性问题的解决方法是使用工具变量。从模型的架构上来看，这个方法是将简单的模型连接起来，组成复杂的“组装”模型。这就是所谓的联结主义，也是后面章节讨论的重点。

本书截止到本章讨论的重点是传统的统计学模型，也就是所谓的数据模型（data model）。这类模型的稳定性和可解释性都很好，但在实际中，单独使用这些模型的预测效果并不理想。为了弥补这个缺憾，之后的章节将集中讲解另一类模型，也就是所谓的算法模型（algorithm model），也称为机器学习。这类模型的数量众多，是人工智能的核心内容。它们往往有非常好的预测效果，但是稳定性和可解释性相对比较薄弱。

仅供非商业用途或交流学习使用

第8章

监督式学习：目标明确

Prediction, not narration, is the real test of our understanding of the world.

（预测，而非叙述，将检验我们对世界的理解。）

—— *Nassim Nicholas Taleb*

8.1 支持向量学习机

8.2 核函数

8.3 决策树

8.4 树的集成

8.5 本章小结

仅供非商业用途或交流学习使用

之前章节讨论的重点是所谓的数据模型（data model）。它们是传统的统计模型，侧重于分析和解释已有数据。从本章开始，我们将进入一个全新的模型世界——算法模型（algorithm model）。这类模型是机器学习以及更广义的人工智能的核心内容。在学术上，算法模型通常分为两类，一类是没有标签数据的非监督式学习，另一类是带有标签数据的监督式学习。本章将讨论后者。

监督式学习的主要目的是利用模型对未知数据做预测。同数据模型类似，监督式学习根据所用数据的不同又可以细分为两类：若标签变量表示类别（变量是离散的），则称为分类；若标签变量表示数值（变量是连续的），则称为回归。从数学的角度上来讲，这两类问题往往是相通的。一方面，可以通过数据分段，将连续标签变量转换为离散标签变量，那么回归问题就变成了分类问题。另一方面，大多数分类模型都是先对数据属于某一类别的概率（或数据到某类别边界的距离）做回归，再基于回归结果得到最终的分类预测。换句话说，分类模型本质上是回归模型。因此，大多数的监督式学习能同时解决分类和回归这两种问题。在实际生产中，分类问题出现得更多，所以分类模型一直是学术研究的重点，也是本章将着重讨论的。

监督式学习的模型很多，无法一一介绍，因此，本章选取了其中具有代表性的、实用性也很强的 3 类模型：支持向量学习机、决策树和树的集成。这 3 类模型的特点分别如下：

- 支持向量学习机的分类效果很好、适用范围也很广，曾一度是机器学习的主流方向，但模型的可解释性一般。
- 决策树模型的建模思路是模拟人在现实生活中做决策的过程，因而模型非常容易理解。但它的预测效果一般，常常需要联结其他模型一起使用。
- 树的集成建立在决策树的基础上，通过使用集成方法提升模型的预测效果，这其中包括随机森林和 GBTs。随机森林的思路是模拟民主投票的过程，借助“集体智慧”解决问题，而 GBTs 的思路是不断地从错误中学习，并以此修正预测结果。这两种模型都能有效地提升模型效果，但模型的可解释性相比于决策树下降了不少。

8.1 支持向量学习机

支持向量学习机（Support Vector Machine，SVM）是一款非常强大的分类模型[1]。它曾经是最好的（不少人相信它现在仍是最好的）监督式机器学习模型，甚至被称为“万能分类器”。在

[1] 这个说法在学术上并不完全准确。最初的支持向量学习机的确是解决分类问题的模型。但在它的基础上拓展出了能解决回归问题的模型，也就是所谓的 SVR（Support Vector Regression），其中最常见的有两种：epsilon-SVR 和 nu-SVR。这些模型在学术上也属于支持向量学习机这个大类，但它们的应用并不如 SVM 广泛，也不在本书的讨论范围内。

仅供非商业用途或交流学习使用

数学上，支持向量学习机的模型处理较为复杂，涉及的知识也较为深奥，常常让人觉得难以理解。因此，本节先略过这些难以理解的数学推导，从直观上和实际操作上来讨论这个模型。

与第 5 章中讨论逻辑回归模型不同，本节只讨论如何用支持向量学习机模型解决二元分类问题。对于多元分类问题，支持向量学习机常使用 one-vs.-all 策略将其拆分为多个二元分类问题解决，具体的细节请参考 5.4.2 节，本章就不再赘述了。

8.1.1 直观例子

先来看一个简单的例子。假设面对的二元分类问题有两个自变量，分别为x_1和x_2。将数据集里的数据表现在图上，可以得到图 8-1，其中圆点表示类别 1，三角形表示类别 0。

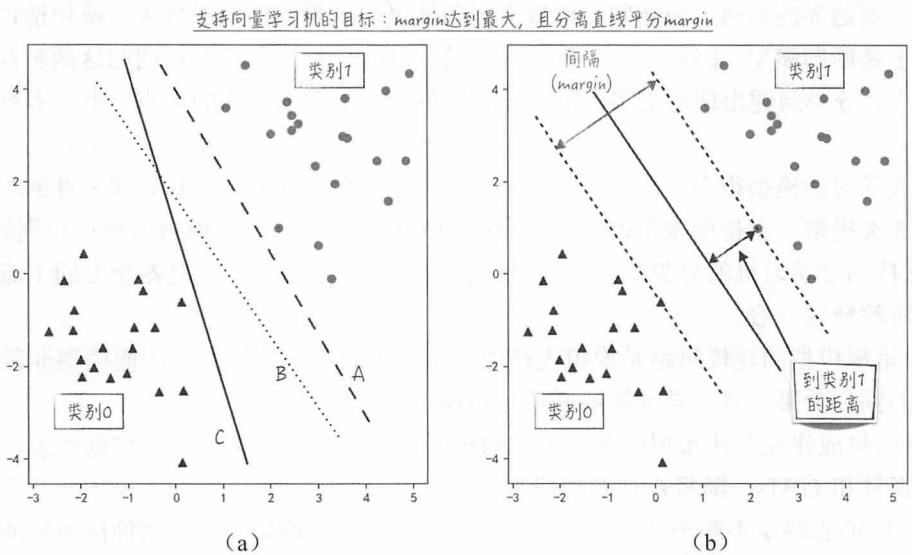


图 8-1

从图像上来看，这两个类别能够被一条直线完美地分开，比如图 8-1a 中的 A、B、C 三条直线。这样的数据在学术上被称为线性可分（linearly separable），而分开它们的直线叫作分离超平面（separating hyperplane）[2]。

如果将直线 A 上方的数据预测为类别 1，直线 A 下方的数据预测为类别 0，可以得到一个百分之百准确的分类模型。对直线 B、C 结论也同样成立。那么，哪一条直线代表的是更加完美的模型呢？从直观上来讲，直线 B 所代表的模型更好，因为它跟每个数据点都离得比

[2] 线性可分的学术定义是，不同类别的数据能被一个超平面（hyperplane）完全分开。表现在二维空间里就是数据能被一条直线分开，表现在三维空间里就是数据能被一个平面分开，以此类推。

仅供非商业用途或交流学习使用

较远，而且离两个类别的距离大致相同。换句话说，直线 B 的倾斜度更加合理，而且位置更加“居中”。相比之下，其他两条直线的倾斜度显得不太合理，而且直线 A 与类别 1 贴得太近，直线 C 又距离类别 0 太近。

除了图 8-1a 中画出的 A、B、C 三条直线外，其实还存在着无数条直线可以将不同类别的数据完全分开。为了从中选出最优的直线，需要将上面提到的倾斜度和位置“居中”做更加量化的解释。

在解决这个问题之前，先来看看数据点到分离直线的距离代表了什么。从之前的讨论可知，作为分离超平面的直线其实是解决分类问题的标准线。落在它上方的点被归为一类，落在它下方的点被归为另一类。那么极端一些，如果某个数据点刚好落在直线上，那么它被归为任何一类都是同样有道理的。因此在支持向量学习机模型里，一个数据点离分离直线的距离代表了模型对这一点预测结果的“自信程度”（在学术上称为置信度），离得越近，模型对预测结果越没有把握。这个结论其实有比较严谨的数学依据，具体的细节将在 8.1.2 节中详细讨论。

基于上面对距离（点到直线）的讨论，支持向量学习机会按照如下的两个原则寻找最优的分离直线。

- 对于一条分离直线，定义它到某一类别的距离等于这个类别中的所有点到这条直线距离的最小值。比如对于图 8-1b 中的实线，虚线（它与实线平行）经过的圆点到实线的距离就是分离直线到类别 1 的距离。支持向量学习机想要最大化分离直线到两个类别的距离之和，在学术上称之为间隔（margin），如图 8-1b 所示。值得注意的是，一旦分离直线的倾斜度确定了，间隔其实也就确定了。因此，最大化间隔这个目标就确定分离直线的倾斜度。
- 在默认情况下，所有数据点的权重都是一样的。所以分离直线到每个类别的距离应该相等。因此如图 8-1b 所示，分离直线应该在两条平行虚线的中间，这样就确定了分离直线的位置。

8.1.2 用数学理解直观

本节的任务是将上面讨论的直观原则转换为严谨的数学定义。在开始这项艰巨的任务之前，先来简单回顾一下有关线性几何的基础知识，这些数学知识是理解后面讨论的关键。需要注意的是，在支持向量学习机的数学讨论中，我们将大量使用向量的内积运算（dot production）。这与前面章节讨论过的线性回归（第 4 章）和逻辑回归（第 5 章）有所不同，这两个模型的数学推导主要使用向量的矩阵乘法。在数学上，矩阵乘法与欧式空间内的向量内积[3]在本质上是

[3] 欧式空间（Euclidean space）是一个很深刻的数学概念，同时日常生活中又会经常遇到它，比如现实世界就是三维欧式空间。对于 n 维向量 $\boldsymbol{a} = (a_1, a_2, \ldots, a_n), \boldsymbol{b} = (b_1, b_2, \ldots, b_n)$，它们的内积定义为 $\boldsymbol{a} \cdot \boldsymbol{b} = \sum_{i=1}^{n} a_i b_i$。更多的细节请参考第 3 章。

仅供非商业用途或交流学习使用

一样的，也可以互相推导。前面章节使用矩阵乘法来推导模型，是为了和学术主流表达方式保持一致，以免读者在阅读其他相关文献时，由于数学符号的不同而感到不解。

先从最直观的二维线性空间谈起，如图 8-2 所示。对于二维空间中的一条直线，假设原点到这条直线的投影为向量$\boldsymbol{\beta}$，也就是说向量$\boldsymbol{\beta}$与直线垂直。那么，对于直线上的任意一点T，向量$\boldsymbol{T}-\boldsymbol{\beta}$与向量$\boldsymbol{\beta}$的内积等于 0。因此，直线可以用一个线性方程来表示，具体的公式如下：

$$\boldsymbol{\beta}\cdot(\boldsymbol{T}-\boldsymbol{\beta})=0 \tag{8-1}$$

反过来，对于任意一个二元线性函数，不妨设为$f(\boldsymbol{X})=\boldsymbol{\gamma}\cdot\boldsymbol{X}$，则二元线性方程$f(\boldsymbol{X})=0$在空间中表示一条直线。事实上，假设点$X_0$使得$f(\boldsymbol{X_0})=0$，对于满足$f(\boldsymbol{X})=0$的任意一点$X$，公式（8-2）成立。

$$\boldsymbol{\gamma}\cdot(\boldsymbol{X}-\boldsymbol{X_0})=0-0=0 \tag{8-2}$$

也就是说，向量$\boldsymbol{X}-\boldsymbol{X_0}$与向量$\boldsymbol{\gamma}$垂直，这表明所有的点$X$都在同一条直线上。而且显然这条直线上的所有点也都满足$f(\boldsymbol{X})=0$，所以任意一个二元线性方程在空间中表示一条直线。

现在再次回到直线，为了表示简单，不妨设它的代数表示为公式（8-1）。那么空间中的任意一点到这条直线的垂直距离应该如何表示呢？分两种情况来讨论，如图 8-2 所示。

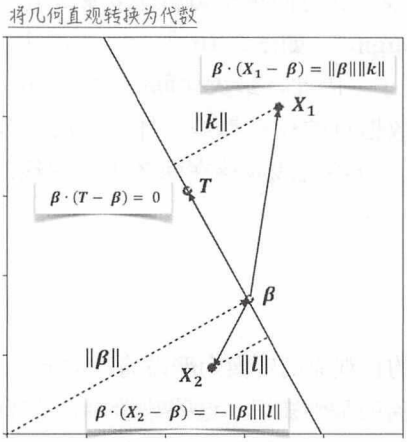


图 8-2

- 如果点在直线上方[4]，比如X_1，假设它到直线的距离为$\|\boldsymbol{k}\|$，那么根据内积的定义，可以得到：

[4] 这样的表述并不严谨。在一般情况下，严谨的表述为：若点在直线上的投影到点的向量与向量$\boldsymbol{\beta}$同方向，则公式（8-3）成立；若点在直线上的投影到点的向量与向量$\boldsymbol{\beta}$反方向，则公式（8-4）成立。

仅供非商业用途或交流学习使用

$$\boldsymbol{\beta} \cdot (\boldsymbol{X}_1 - \boldsymbol{\beta}) = \|\boldsymbol{\beta}\|\|\boldsymbol{k}\| \cos 0 = \|\boldsymbol{\beta}\|\|\boldsymbol{k}\| \tag{8-3}$$

- 如果点在直线下方，比如X_2，假设它到直线的距离为$\|\boldsymbol{l}\|$，那么类似地可以得到：

$$\boldsymbol{\beta} \cdot (\boldsymbol{X}_2 - \boldsymbol{\beta}) = \|\boldsymbol{\beta}\|\|\boldsymbol{k}\| \cos \pi = -\|\boldsymbol{\beta}\|\|\boldsymbol{l}\| \tag{8-4}$$

将上面的讨论总结一下，对于一个给定的线性方程$f(\boldsymbol{X}) = \boldsymbol{\gamma} \cdot \boldsymbol{X}$，绝对值$|f(X)|$与点$X$到直线$f(\boldsymbol{X}) = 0$的距离成正比，而$f(\boldsymbol{X})$的符号表示点$X$到直线的垂线方向。这个结论不仅仅针对二维空间，在高维空间同样成立。只不过在高维空间里，直线要变成超平面[5]。

有了上面的代数知识，让我们稍稍离题一下，再次讨论第 5 章介绍的逻辑回归模型。逻辑回归模型同样也是一个分类模型，不妨设模型所用的自变量为$\boldsymbol{X}$，需要预测的因变量为$\boldsymbol{Y}$，其中，$\boldsymbol{Y} = 1$表示一类，$\boldsymbol{Y} = 0$表示另一类，则它的预测公式如下：

$$P(\boldsymbol{Y} = 1) = {}^{1}/_{(1 + \mathrm{e}^{-X\beta})} \tag{8-5}$$

公式（8-5）中的线性部分记为函数$f(\boldsymbol{X}) = \boldsymbol{X\beta}$，当$f(\boldsymbol{X}) > 0$时，$f(\boldsymbol{X})$越大，模型预测$\boldsymbol{Y} = 1$的概率就越接近于 1。根据上面的讨论可以知道，当$f(\boldsymbol{X}) > 0$时，$f(\boldsymbol{X})$与$X$到超平面$f(\boldsymbol{X}) = 0$的距离成正比。这用几何直观的语言表示出来就是：若一个点在超平面正向一侧，它离超平面越远，逻辑回归模型就越有把握预测它的标签等于 1。类似地，可以得到当$f(\boldsymbol{X}) < 0$时的几何直观解释：若一个点在超平面反向一侧，它离超平面越远，逻辑回归模型就越有把握预测它的标签等于 0。

将上述内容总结一下，从几何直观上，逻辑回归模型的分类原理可描述为：模型的线性部分$\boldsymbol{X\beta} = 0$定义了分类的分离超平面，离这个超平面越远，模型对相应的预测结果把握越大。而这与 8.1.1 节中讨论的支持向量学习机的分类思路非常相近。因此，支持向量学习机可以被看作逻辑回归的一种“基因突变”。虽然两者外表看起来相差很大，但其数学本质在很大程度上是一致的。

8.1.3 从几何直观到最优化问题

现在回来看在数据线性可分的情况下，用数学语言描述支持向量学习机模型，或者讲得直接一些，我们将找出这个模型对应的最优化问题。在下面的讨论中，假设训练数据i的自变量为$\boldsymbol{X}_i$，被预测量为y_i。

如图 8-3 所示，分离直线记为$\boldsymbol{\beta} \cdot \boldsymbol{X} + b = 0$，而虚线穿过的点分别表示两者类别中到分离直线最近的点，记为点m和点n。根据模型分类的第二点原则，直线是居中的，因此m和n到分离直线的距离都为$\|\boldsymbol{l}\|$。

同公式（8-3）和公式（8-4）类似，图 8-3 中的两边虚线分别可以表示为$\boldsymbol{\beta} \cdot \boldsymbol{X} + b = \|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$和$\boldsymbol{\beta} \cdot \boldsymbol{X} + b = -\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$。对于类别 1 中的任意一点$X_i$，它到分离直线的距离大于

[5] 事实上，直线就是二维空间的超平面，所以这只是表述习惯的问题。

仅供非商业用途或交流学习使用

$\|\boldsymbol{l}\|$，因此，$\boldsymbol{\beta}\cdot\boldsymbol{X}_i+b\geqslant\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$。类似地，对于类别 0 中的任意一点$X_i$，它到分离直线的距离也大于$\|\boldsymbol{l}\|$，而且它在直线的下方，因此，$\boldsymbol{\beta}\cdot\boldsymbol{X}_i+b\leqslant-\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$。

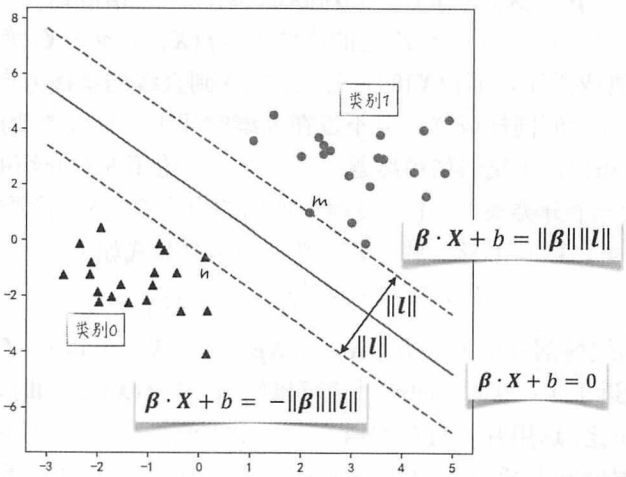


图 8-3[6]

支持向量学习机的目的是最大化两条虚线之间的距离（也就是之前提到的 margin）。那么模型的最优化问题可以表示为公式（8-6）。

$$\begin{gathered}\max 2\|\boldsymbol{l}\|\\ \text{任一}\boldsymbol{X}\text{属于类别 1，}\ \boldsymbol{\beta}\cdot\boldsymbol{X}+b\geqslant\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|\\ \text{任一}\boldsymbol{X}\text{属于类别 0，}\ \boldsymbol{\beta}\cdot\boldsymbol{X}+b\leqslant-\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|\end{gathered}\tag{8-6}$$

为了后面的计算更加方便，记$\boldsymbol{w}=\boldsymbol{\beta}/\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$，$c=b/\|\boldsymbol{l}\|\|\boldsymbol{\beta}\|$，可以得到$2\|\boldsymbol{l}\|=2/\|\boldsymbol{w}\|$。因此将公式（8-6）改写为更为标准的公式（8-7）。之后的讨论都将基于后者。

$$\begin{gathered}\max\frac{2}{\|\boldsymbol{w}\|}\\ \text{任一}\boldsymbol{X}\text{属于类别 1，}\ \boldsymbol{w}\cdot\boldsymbol{X}+c\geqslant 1\\ \text{任一}\boldsymbol{X}\text{属于类别 0，}\ \boldsymbol{w}\cdot\boldsymbol{X}+c\leqslant-1\end{gathered}\tag{8-7}$$

同时，我们用$y_i=1$表示类别 1，$y_i=-1$表示类别 0[7]。那么在公式（8-7）中的限制条件两边同时乘以y_i，可以将两条限制条件合并为一条。同时注意到，$\max 2/\|\boldsymbol{w}\|$等价于$\min\frac{1}{2}\|\boldsymbol{w}\|^2$。这样就得到了支持向量学习机最简洁也是最常用的数学表示（在数据线性可分的情况下）。

[6] 图像的代码实线请参考随书配套的代码/ch08-supervised/svm/linear_separable_svm.py。

[7] 支持向量学习机中，被预测量y_i的赋值与逻辑回归模型中的不一样。在逻辑回归中，用$y_i=0$表示类别 0，$y_i=1$表示类别 1。这两者赋值方式对模型结果没有任何影响，纯粹只是为了数学推导的方便。

仅供非商业用途或交流学习使用

$$\min \frac{1}{2}\|\boldsymbol{w}\|^2$$
$$y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \geqslant 1 \tag{8-8}$$

求解公式（8-8）所描述的最优化问题，就可以得到支持向量学习机模型参数的估计值，不妨设为$\hat{w}, \hat{c}$。模型的预测公式如下，其中，$\hat{y}_i$ 表示模型的预测结果，而sign表示数字的符号，当$x > 0$时，$\text{sign}(x) = 1$；否则$\text{sign}(x) = -1$。

$$\hat{y}_i = \text{sign}(\hat{\boldsymbol{w}} \cdot \boldsymbol{X_i} + \hat{c}) \tag{8-9}$$

值得注意的是，公式（8-8）是带有限制条件的最优化问题（限制条件为$y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \geqslant 1$）。这和之前接触到线性回归和逻辑回归不同，它们对应的最优化问题是没有限制条件的，比如针对线性回归模型，参数估算公式为$\hat{\boldsymbol{\beta}} = \text{argmin} \sum_i (y_i - \boldsymbol{X_i}\boldsymbol{\beta})^2$。

在数学上，求解有限制条件的最优化问题比没有限制条件的更加困难，所用的数学工具也有所差异。具体的细节将在 8.2 节中讨论。

8.1.4 损失项

到目前为止，我们都在数据线性可分的情况下，讨论支持向量学习机的分类方法。但如果仅限于此，支持向量学习机这个模型几乎没有实用价值。因为在现实生活中，需要处理的数据集绝大多数都是线性不可分的，也就是说不存在一个超平面使得不同类别的数据分别落在平面的不同两侧。以图 8-4 为例，图中的点a是类别 0，但它更靠近类别 1 的中心。针对这份数据，就没有办法像之前一样，用一条直线将两个类别完全分开。那在这种情况下，支持向量学习机会如何处理呢？答案就是加入损失项。

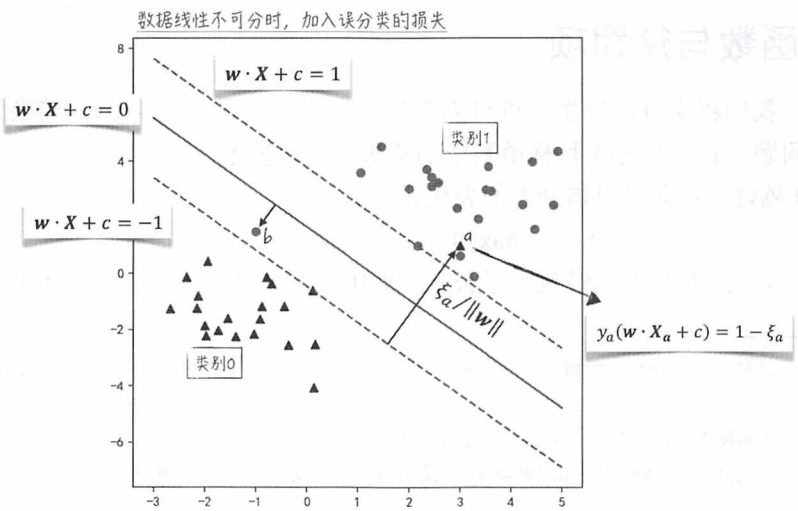


图 8-4

书籍是人类进步的阶梯

在数学上，支持向量学习机通过公式（8-8）中的条件$y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \geqslant 1$来保证分类结果的准确。比如在图 8-4 中，这个条件将保证数据表示的点都落在两条虚线的外面，这也是模型的分类假设。现在由于数据线性不可分，所以将这个条件稍稍放松一点，改为$y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \geqslant 1 - \xi_i$。其中$\xi_i \geqslant 0$，它与点$i$离相应虚线的距离成正比。比如图 8-4 中点$a$，可以计算得到它到下方虚线的距离为$\xi_a/\|\boldsymbol{w}\|$，到分离超平面（实线）的距离为$(\xi_a - 1)/\|\boldsymbol{w}\|$。需要注意的是，这里计算的距离是空间的实际欧式距离。但在机器学习中（比如第三方模型库 scikit-learn），针对一个支持向量学习机模型，常把函数值$\xi_a - 1 = \boldsymbol{w} \cdot \boldsymbol{X_a} + c$（注意到图中的$a$点有$y_a = -1$）称为数据到分离超平面的“距离”[8]。

因此，ξ_i可以被看作模型在数据i这一点违反自身分类原则的程度，也就是模型在这一点的损失[9]。那么它们的和$\sum_i \xi_i$当然是越小越好。这个目标和模型本身的目标（最大化 margin）相矛盾，比如在图 8-4 中，模型原本的目标是最大化两条虚线之间的距离，但如果两条虚线离得越远，显然，模型“错误”的损失$\sum_i \xi_i$也越大。也就是说，从数学上来看函数$\frac{1}{2}\|\boldsymbol{w}\|^2$和函数$\sum_i \xi_i$是负相关关系，就像跷跷板的两头一样。

为了兼顾上面提到的两个模型目标，我们通过线性函数来折衷。具体地，支持向量学习机的最优化问题被改为公式（8-10）。其中，$C > 0$为模型的损失系数，是一个超参数，它对模型结果的影响将在 8.1.6 节中讨论。

$$\begin{gathered}\min \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \\ y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \geqslant 1 - \xi_i;\ \xi_i \geqslant 0\end{gathered} \tag{8-10}$$

在实际应用中，不管数据是否线性可分，都会根据公式（8-10）来估计支持向量学习机的模型参数。公式（8-8）更多地只是为了展示模型的建模思路。

8.1.5 损失函数与惩罚项

接下来，我们将支持向量学习机对应的最优化问题（公式（8-10））转换为不带限制条件的最优化问题。首先只考虑目标函数中的损失部分：$\sum_i \xi_i$。在不影响其他参数的情况下，ξ_i越接近于 0 越好。由此可以得到ξ_i的表达式：

$$\xi_i = \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c)) \tag{8-11}$$

事实上，公式（8-11）已经包含了公式（8-10）中的限制条件。理由如下：

---

[8] 在数学上，这样定义的函数值的确是一种距离，只不过它不是我们通常熟悉的欧式距离。实际上它等于欧式距离乘以一个常数$\|\boldsymbol{w}\|$。

[9] 事实上，ξ_i作为模型“错误”的损失项可以被分为两个层次：

- 分类错误，也就是模型最终的预测结果错误。从数学上来讲就是$\xi_i > 1$；从图形上来看，就是数据点落在了实线的另一侧。
- 分类结果违反假设。这种情况下，模型最终的预测结果虽然是正确的，但是数据到分离超平面的距离不够远。这种情况在数学上表现为$0 \leqslant \xi_i \leqslant 1$；在图形上表现为数据点落在实线和虚线之间。

书籍是人类进步的阶梯

$$\xi_i = \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c)) \geqslant 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c) \tag{8-12}$$

那么支持向量学习机的参数估计公式可改写为，其中，$C > 0$是模型给定的超参数。

$$\min_{w,c} \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c)) \tag{8-13}$$

公式（8-13）其实定义了支持向量学习机的损失函数：$L = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c))$，将它分为如下两部分。

- 模型的预测损失$LL = \sum_i \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X_i} + c))$，学术上称这个函数为 hinge loss。由此可以看到支持向量学习机的预测损失，其实是由一个线性损失函数外套一个非线性变换构成的。这再一次印证了本书在前面章节重复过很多次的一个观点：复杂模型都是由线性模型和非线性变换一层层嵌套构成的。在支持向量学习机中，非线性变换部分$f(x) = \max(0, x)$是所谓的线性整流函数（ReLU）。它是深度学习中非常重要的一个函数，我们将在后面的章节中详细讨论这个函数的细节。
- 惩罚项$\frac{1}{2}\|\boldsymbol{w}\|^2$。为了解决模型过拟合的问题，在第 4 章中讨论了惩罚项这个概念，而这里的$\frac{1}{2}\|\boldsymbol{w}\|^2$正是 L2 惩罚项。虽然在支持向量学习机中，这一项的初衷是最大化 margin，使得分离超平面尽量远离边缘数据点，但这样会在分类分界面和数据之间留出一道空隙，在某种程度上的确防止了模型出现过拟合，因此符合惩罚项的定义。与其他模型不同的是，支持向量学习机的惩罚项是并不可少的，而其他模型在理论上并没有这样的要求（虽然在实际应用中，不管使用什么模型，在损失函数中加入惩罚项几乎是一定的）。

综合上面的分析，支持向量学习机的场景类型和损失函数如图 8-5 所示。

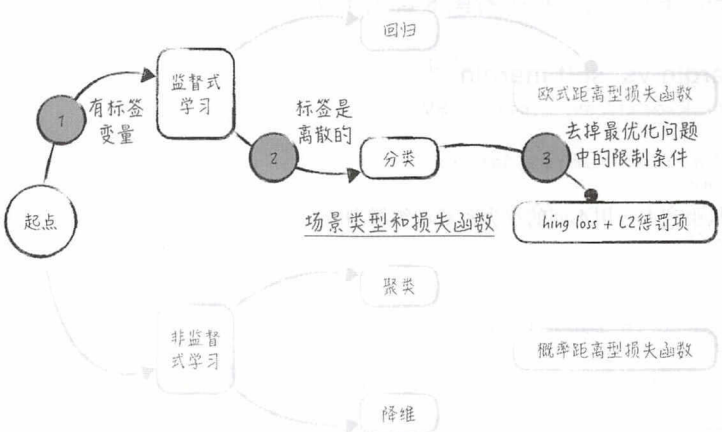


图 8-5

8.1.6 Hard margin 与 soft margin 比较

在支持向量学习机的损失函数中，有一个很重要的超参数——模型的损失系数C。这

书籍是人类进步的阶梯

个超参数的取值将极大地影响模型的分类结果。为了说明这个问题，再来看看模型的损失函数：

$$L = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \max(0, 1 - y_i(\boldsymbol{w} \cdot \boldsymbol{X}_i + c)) \tag{8-14}$$

超参数C其实是模型预测损失的权重[10]，它的值很大表示模型将更加在意分类“错误”而轻视 margin 的大小。下面用一个具体的例子来展示这种影响，具体实现如程序清单 8-1 所示。

（1）公式（8-14）所表示的模型其实是线性支持向量学习机（非线性支持向量学习机将在 8.2 节中讨论）。对于这种模型，第三方库 scikit-learn 提供了两种实现方式：一种是 sklearn.svm.SVC，如第 1 行代码所示；另一种是 sklearn.svm.LinearSVC。虽然使用后一种实现，可以为线性 SVM 选择其他种类的损失函数和惩罚项[11]，但在实际中，更加常用的是第一种实现（因为可以方便地更换核函数）。

（2）类 SVC 里有参数“C”，这个参数就是公式（8-14）里的C。因此如第 11 行代码所示，对同一数据集使用不同的损失系数C训练模型，然后将这些不同模型的结果可视化。值得注意的是，SVC 里的参数“kernel”表示支持向量学习机所使用的核函数，这里使用“linear”（线性核函数）。有关核函数的细节请参考 8.2 节。

（3）对于训练好的模型“re”，它有一个函数：decision_function。将数据i的自变量部分（即X_i）传给它，可以得到该数据到分离超平面的“距离”。但值得注意的是，正如 8.1.4 节里提到的，这里所指的“距离”并非空间中的欧式距离，而是函数值$\boldsymbol{w} \cdot \boldsymbol{X}_i + c$。因此，“decision_function=±1”刚好对应着图 8-6a 中的虚线。

程序清单 8-1 hard margin vs. soft margin[12]

```
 1 | from sklearn.svm import SVC
 2 |
 3 | def hardSoftMargin(data):
 4 |     """
 5 |     从小到大，用不同的损失系数训练模型
 6 |     """
 7 |     C = []
 8 |     res = []
 9 |     for i in range(-1, 5):
10 |         C_ = 10 ** i
11 |         model = SVC(C=C_, kernel="linear")
```

[10] 对于之前讨论的其他模型，我们经常接触到另一个超参数：惩罚项权重（具体细节可参考第 4 章）。事实上超参数C其实就是惩罚项权重的倒数（由于损失函数的具体值并不影响模型的参数估计，所以对公式（8-13）除以C就可以得到结论），它的值越小表明惩罚的力度越大。

[11] 在具体的实现上，LinearSVC 提供两种损失函数：一种是本书讨论过的 hinge loss，另一种是 squared hinge loss（hinge loss 的平方）。它还提供两种惩罚项：一种是本书讨论过的 L2 惩罚项，另一种是 L1 惩罚项（模型参数绝对值的和）。

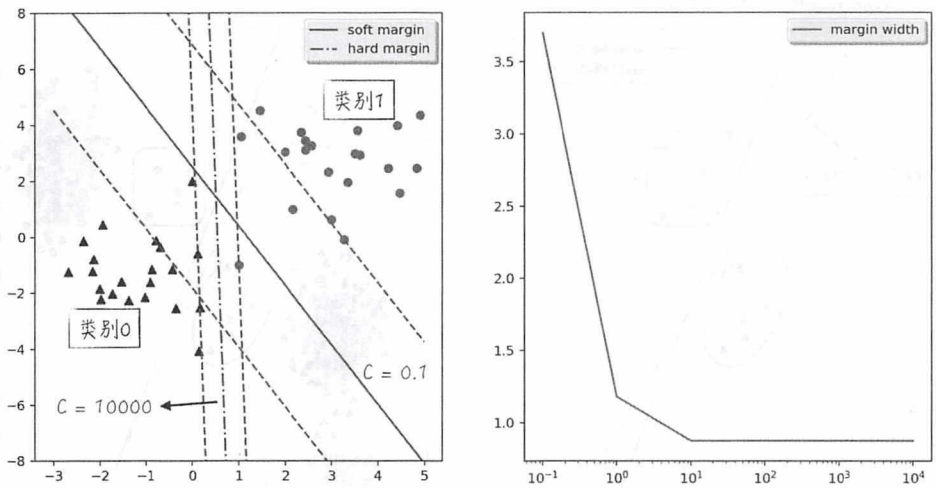
[12] 完整的代码实线请参考随书配套的代码/ch08-supervised/svm/hard_soft_margin.py。

书籍是人类进步的阶梯


```
        model.fit(data[["x1", "x2"]], data["y"])
        res.append(model)
        C.append(C_)
    visualize(data, C, res)
```

运行完整的代码，可以得到如图 8-6 所示的结果。从整体上来看，随着参数C的增大，margin width（两条虚线之间的距离）会减小，如图 8-6b 所示。

从分类结果上来看，当C的取值比较小时，比如$C = 0.1$，有比较多的点落在相应的虚线之间，甚至有的越过了实线，换句话说，模型更注重的是靠近类别中心的数据点。在学术上，这种情况被形象地称为 soft margin；当C的取值比较大时，比如$C = 10\,000$，几乎没有点落在相应的虚线之间，也就是说，模型对靠近分离超平面的“异常点”更加重视。类似地，这种情况在学术上被称为 hard margin，如图 8-6a 所示。



（a）　　（b）

图 8-6

在实际应用中，参数C可以被当作模型的超参数来使用：通过第 4 章中介绍的网点搜寻（grid search）来找到使预测效果最好（使用第 5 章中讨论的分类结果评估指标）的取值。也可以根据实际需要，人为地选择合适的值，比如对于搭建模型的数据，如果已知其中有比较多的随机扰动，那么就应该选择较小的C值，也就是 soft margin，使模型结果少受数据噪声的影响。

8.1.7 支持向量学习机与逻辑回归：隐藏的假设

支持向量学习机和第 5 章中讨论的逻辑回归都是分类模型，那么这两个模型有什么不同

书籍是人类进步的阶梯

呢？在实际应用中，针对一个具体的问题，应该如何选择它们呢？我们先通过一个简单的例子来看这两个模型的差异。

（1）首先生成一批较少的，离得比较近的数据集。如图 8-7a 所示，数据分为两个类别，图形符号与之前的例子保持一致。对这批数据分别使用支持向量学习机和逻辑回归建模，并将得到的模型结果也表现在图上。其中实线表示支持向量学习机的分离超平面，点线表示逻辑回归的分离超平面。

（2）在第（1）步数据集的基础上，增加一批数量比较多，离得比较远的新数据，如图 8-7b 所示。两幅图中用方框圈起来的数据是一样的。针对这个新的数据集，同样分别使用两种模型建模，并将相应结果表现在图上。

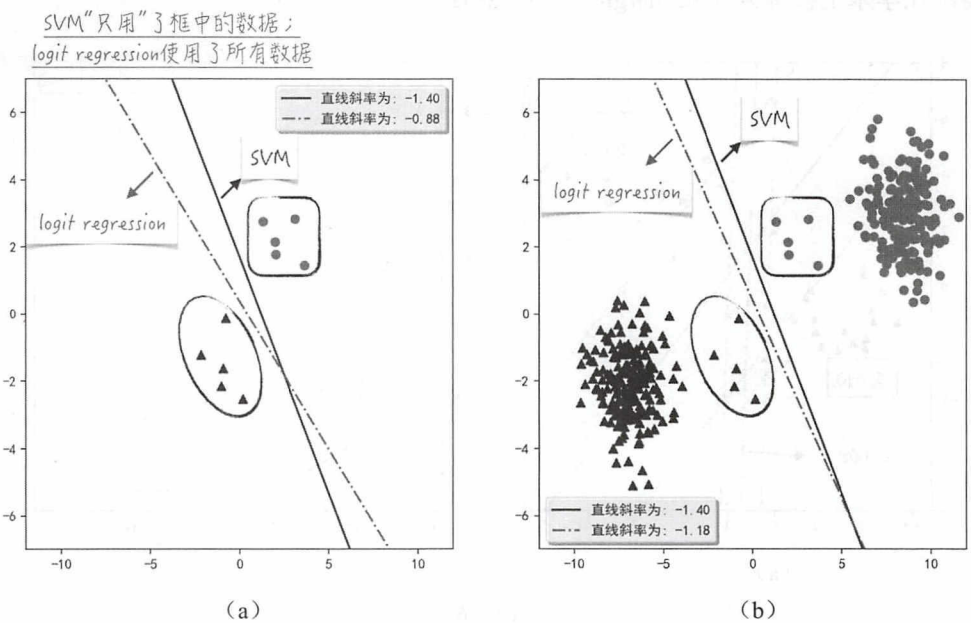


图 8-7[13]

对比图 8-7a 和图 8-7b 中的实线会发现虽然训练模型的数据是不一样的，但支持向量学习机（模型的损失系数$C = 1$）的模型结果是一模一样的。换句话说，新加入的数据对模型结果没有影响。从数据的角度来讲，由于新加入的数据远离分离超平面，所以它们在支持向量学习机这个模型里的权重等于 0。这正是支持向量学习机的隐含假设：每个数据点权重其实并不一样，越靠近分离超平面，数据的权重也就越大。而且损失系数C越大（hard margin），数据权重不一样的现象也就越严重，如图 8-8 所示，hard margin 在两组数据里的模型结果是

[13] 图 8-7 和图 8-8 的代码实现请参考随书配套的代码/ch08-supervised/svm/svm_vs_logit.py。

书籍是人类进步的阶梯

一样的，而 soft margin 的结果则有较大变化。支持向量学习机的这个隐含假设其实有严谨的数学证明，具体细节请参考 8.2.3 节。

对这种模型隐含假设的理解是至关重要的。毫不夸张地说，这是区分优秀数据科学家的标准之一。这种隐含假设与模型的其他假设不同，在搭建模型时并没有被明确地提出，所以往往被大家忽略，但它们对模型结果的影响又是巨大的。建模时，如果忽略或者没能正确地处理这种假设，会导致模型结果较差甚至完全错误，而且会让人产生错误的幻觉：数据满足模型的每一个假设，但模型的效果就是不好。

而逻辑回归则没有这样的隐含假设，除非特殊处理[14]，模型对每个数据点的权重都是一样的。因此当加入新数据后，模型的结果发生了比较大的改变，如图 8-7a 和图 8-7b 中的点线所示。

总结一下，这两个模型对数据权重的隐含假设并不相同。这也给了我们选择模型的一个依据，如果希望模型对数据中靠近“边缘”的点更加敏感，则推荐使用 hard margin 支持向量学习机。如果需要综合考虑每一个点，则需要使用逻辑回归或者 soft margin 支持向量学习机。

soft margin使用了所有数据；
hard margin"只用"了框中的数据

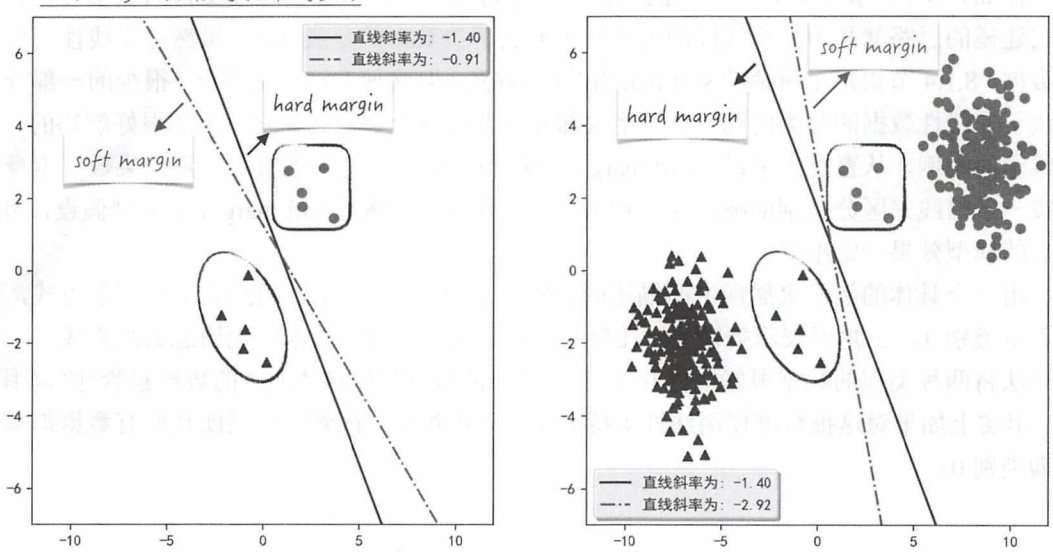


图 8-8

[14] 在处理非均衡数据集时，通常需要对不同的类别赋予不同的权重。具体的细节可参考 5.5 节。

书籍是人类进步的阶梯

8.2 核函数

核函数（kernel function 或者 kernel trick）是机器学习中十分重要的数据处理技巧。通过核函数，我们可以将原空间中非线性的数据映射为高维或无限维空间中近似线性关系的数据。既然在新的空间里，数据是近似线性关系，那么就可以使用线性模型对其建模分析。这和本书一直强调的建模思路是一致的：通过某种数学变换，将非线性问题转换为线性问题解决。

在实际生产中，核函数常常结合支持向量学习机一起使用，使得模型能够解决各类非线性分类问题。当然，几乎所有的机器学习模型都可以搭配核函数使用，比如线性回归模型加上核函数得到所谓的核岭回归（Kernel Ridge Regression，KRR）[15]。虽然机器学习的模型繁多，差异很大，但它们搭配核函数的方法大体上是一致的，因此本节以支持向量学习机为例，讨论核函数的使用。

8.2.1 空间变换：从非线性到线性

在 8.1 节中，我们讨论了经典的支持向量学习机。它也被称为线性支持向量学习机，因为它建模的思路就是用一个线性的分离超平面去解决数据的分类问题。虽然对于线性不可分的数据，8.1.4 节提供了所谓的 soft margin 解决方案，但这种方法只是部分（很少的一部分）解决了非线性数据的分类问题。对于绝大部分的非线性分类问题，它是无法很好解决的。以二维空间为例，从直观上来讲，soft margin 的模型结果依然是一条直线。那如果数据本身就是按一条曲线来区分类别的呢？在这种情况下，数据不再满足 soft margin 的模型假设，因此最后的模型效果一定不好。

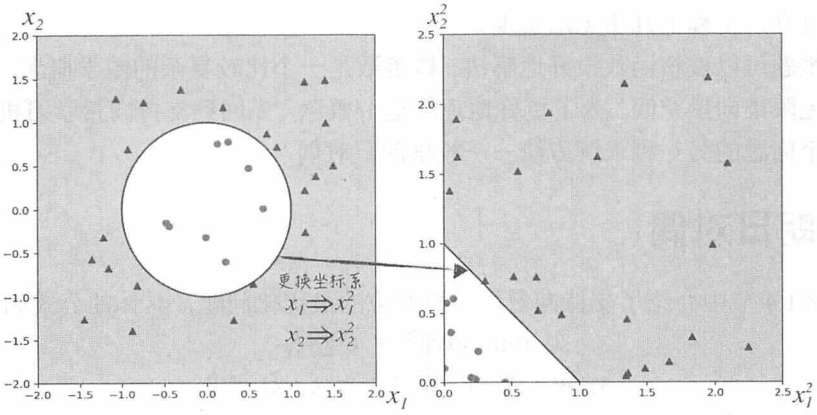
用一个具体的例子来展现上面描述的问题。如图 8-9a 所示，沿用之前的表示方式，圆点表示类别 1，三角形表示类别 0。直观上，对于图中表示的数据，无论怎么画直线，都没有办法将两种类别的数据很好地分开，因为类别 1 的数据好像被类别 0 的数据紧紧地“包围”住。事实上如果对这批数据使用线性支持向量学习机模型，得到的结果就是所有数据都被预测为类别 0。

---

[15] 在 4.3.3 节中讨论惩罚项时，我们引入了 L2 惩罚。带有 L2 惩罚项的线性回归就叫岭回归（ridge regression），而核岭回归就是使用岭回归对新数据（经过核函数变换后的数据）进行回归建模。

值得注意的是，在统计的非参模型（nonparametric model）中存在另一种回归和核函数的结合：核回归（kernel regression）。这个模型虽然和核岭回归的名字很接近，但建模思路和模型形式完全不同，是与后者几乎没有共同点的一个模型。

书籍是人类进步的阶梯



(a) (b)

图 8-9

但如果换个角度，在图中画一个以原点为中心，半径为 1 的圆圈，则刚好可以很完美地将两个类别的数据分开，如图 8-9a 所示，三角形全部落在圆圈外的灰色区域。这么看来，为了解决这个问题，需要重新设计一款“圆圈版”的支持向量学习机，但事情并没有想象中那么复杂。从空间变换的角度重新梳理一下“画圆圈”这件事情，假设图 8-9a 的横纵坐标为x_1, x_2，如图 8-9b 所示，将坐标系的横坐标改为x_1^2、纵坐标改为x_2^2。那么图 8-9a 中的圆圈就变成了图 8-9b 中的直线，而圆圈内的圆点会落在新坐标系下的左下角三角区域内。这样，原本非线性的分类问题在新的坐标系下就变成了线性的分类问题，而后者可以用支持向量学习机完美地解决。将上面的整个过程抽象为严格的数学语言。

- 假设分类问题的原始数据为$\{\boldsymbol{X}_i, y_i\}$，其中，$\boldsymbol{X}_i$为自变量，$y_i$为被预测量。
- 找到一个非线性的空间变换$\boldsymbol{\phi}$，将数据转换为$\{\boldsymbol{\phi}(\boldsymbol{X}_i), y_i\}$。比如上面的例子中，$\boldsymbol{X}_i = (x_{1,i}, x_{2,i})$、$\boldsymbol{\phi}(\boldsymbol{X}_i) = (x_{1,i}^2, x_{2,i}^2)$。
- 使用转换后的新数据训练模型，并得到如下的预测公式，它与公式（8-9）（线性支持向量学习机的预测公式）非常相似。

$$\hat{y}_i = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{X}_i) + c) \tag{8-15}$$

这个方法在理论上很完美，但实际上很难直接使用，原因主要有两个。首先，很难得到转换函数$\boldsymbol{\phi}$的具体表达式。在上面的例子中，数据只是二维的，我们通过将其可视化才联想到相应的转换函数。在实际产生中，数据是高维的，很难将其可视化。即使能将数据直观地呈现出来，也很难从中联想到要如何做高维空间之间的变换。其次，对于非线性分类问题，往往要将原始数据转换到高维空间才能达到近似线性可分的目的（这其实相当于从原始数据中提取更多特征来搭建模型）。因此，如果直接使用$\{\boldsymbol{\phi}(\boldsymbol{X}_i), y_i\}$训练模型，则常常导致模型的

书籍是人类进步的阶梯

运算复杂度过高，工程上几乎无法实现。

这两个难题可以被核函数很好地解决。核函数是一个比较复杂的数学概念，涉及向量空间的内积和无限维向量空间。为了更好地理解这个概念，先回到支持向量学习机的最优化问题，讨论这个问题的另一种求解方法——拉格朗日对偶。

8.2.2 拉格朗日对偶

我们在 8.1.4 节中介绍了支持向量学习机对应的最优化问题，具体的公式如下：

$$\begin{gathered}\min \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \\ y_i(\boldsymbol{w}\cdot\boldsymbol{X_i} + c) \geqslant 1 - \xi_i;\ \xi_i \geqslant 0\end{gathered} \tag{8-16}$$

随后的 8.1.5 节讨论了如何将公式（8-16）转换为没有限制条件的最优化问题，但提供的解决方法并不具备普遍性，也没有触及问题的本质。现在，我们将讨论对于这类问题更加简便也更加工程化的解决方法，这就是在机器学习中应用非常广泛的拉格朗日对偶[16]。当然这个方法涉及很多深刻的数学知识，比较复杂，对数学细节不感兴趣的读者可以跳过其中的数学证明，记住结论即可（见图 8-10）。

将公式（8-16）泛化为更加抽象的最优化问题，如公式（8-17）所示：

$$\begin{gathered}\min f(\theta) \\ g_i(\theta) \leqslant 0, i = 1,\ldots,k \\ h_i(\theta) = 0, i = 1,\ldots,l\end{gathered} \tag{8-17}$$

上面的这个问题在学术上被称为原始问题（primal optimization problem）。基于它定义相应的拉格朗日函数（Lagrangian 或者 generalized Lagrangian），其中，α_i, β_i在学术上被称为拉格朗日乘数（Lagrange multiplier）。

$$L(\theta, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\theta) + \sum_i \alpha_i g_i(\theta) + \sum_i \beta_i h_i(\theta) \tag{8-18}$$

接下来，违反常规将问题搞得“更复杂”。不妨设$P(\theta) = \max_{\alpha \geqslant 0, \beta} L(\theta, \boldsymbol{\alpha}, \boldsymbol{\beta})$。当$\theta$的值满足公式（8-17）中的限制条件时，可以得到$P(\theta) = f(\theta)$；当$\theta$的取值不满足限制条件时，也容易得到$P(\theta) = \infty$。因此，公式（8-17）等价于公式（8-19）这个更复杂的双重最优化问题。

$$\min_\theta P(\theta) = \min_\theta \max_{\alpha \geqslant 0, \beta} L(\theta, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{8-19}$$

现在定义所谓的对偶问题（dual optimization problem）。不妨设$D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_\theta L(\theta, \boldsymbol{\alpha}, \boldsymbol{\beta})$，则公式（8-19）的对偶问题为：

$$\max_{\alpha \geqslant 0, \beta} D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \max_{\alpha \geqslant 0, \beta} \min_\theta L(\theta, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{8-20}$$

[16] 这个方法以法国数学家约瑟夫·拉格朗日（Joseph Lagrange）命名，是一种在一个或多个约束条件下，寻找多元函数极值的方法。

书籍是人类进步的阶梯

在一定条件下（这些条件，当然公式（8-16）都满足）[17]，原始问题和对偶问题是等价的。也就是说，求解公式（8-17）可以转换为求解公式（8-20）。而且更重要的一点，这两个问题的最优解一定存在，并满足所谓的卡罗需-库恩-塔克条件[18]（Karush-Kuhn-Tucker conditions，KKT 条件）。

不妨设$\hat{\theta},\hat{\alpha},\hat{\beta}$是达到最值时的参数值，即$L(\hat{\theta},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}) = \min_{\theta} P(\theta) = \max_{\boldsymbol{\alpha} \geqslant 0,\boldsymbol{\beta}} D(\boldsymbol{\alpha},\boldsymbol{\beta})$，那么 KKT 条件如公式（8-21）所示：

$$\begin{gathered}\frac{\partial L}{\partial \theta}(\hat{\theta},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}) = 0\\ \frac{\partial L}{\partial \beta}(\hat{\theta},\hat{\boldsymbol{\alpha}},\hat{\boldsymbol{\beta}}) = 0\\ \hat{\alpha}_i g_i(\hat{\theta}) = 0\\ g_i(\hat{\theta}) \leqslant 0\\ \hat{\alpha}_i \geqslant 0\end{gathered} \tag{8-21}$$

有了上面的数学基础，现将公式（8-16）转换为相应的对偶问题。需要注意的是，公式（8-16）里只有$g_i(\theta) \leqslant 0$这一类限制条件。首先定义相应的拉格朗日函数：

$$L(\boldsymbol{w},c,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\gamma}) = \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i\left[y_i(\boldsymbol{w}\cdot \boldsymbol{X}_i + c) - 1 + \xi_i\right] - \sum_i \gamma_i \xi_i \tag{8-22}$$

于是对偶函数为$D(\boldsymbol{\alpha},\boldsymbol{\gamma}) = \min_{\boldsymbol{w},c,\boldsymbol{\xi}} L(\boldsymbol{w},c,\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\gamma})$，应用公式（8-21）中的第一个等式可以得到：

$$\begin{gathered}\frac{\partial L}{\partial w} = 0 \Rightarrow \hat{\boldsymbol{w}} = \sum_i \hat{\alpha}_i y_i \boldsymbol{X}_i\\ \frac{\partial L}{\partial \xi} = 0 \Rightarrow C - \hat{\alpha}_i - \hat{\gamma}_i = 0\\ \frac{\partial L}{\partial c} = 0 \Rightarrow \sum_i y_i \hat{\alpha}_i = 0\end{gathered} \tag{8-23}$$

注意到$\hat{\boldsymbol{w}} = \sum_i \hat{\alpha}_i y_i \boldsymbol{X}_i$，于是模型的预测公式可以写为公式（8-24），其中$\boldsymbol{X}_i \cdot \boldsymbol{X}_j$表示两个向量之间的内积。

$$\hat{y}_j = \operatorname{sign}\left(\sum_i \hat{\alpha}_i y_i (\boldsymbol{X}_i \cdot \boldsymbol{X}_j) + \hat{c}\right) \tag{8-24}$$

经过一些比较复杂的计算，可以得到支持向量学习机的对偶形式（公式（8-25）），整个过程如图 8-10 所示。

$$\begin{gathered}\max_{\alpha} \sum_i \alpha_i - \tfrac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j (\boldsymbol{X}_i \cdot \boldsymbol{X}_j)\\ 0 \leqslant \alpha_i \leqslant C;\ \sum_i \alpha_i y_i = 0\end{gathered} \tag{8-25}$$

---

[17] 这些条件列举如下（显然，所列举的条件，支持向量学习机都满足）。

- 函数f和g_i都为凸函数（convex function），即对于任意定义域内的θ_1,θ_2以及$0 \leqslant t \leqslant 1$，以$f$为例，则$f[t\theta_1 + (1-t)\theta_2] \leqslant tf(\theta_1) + (1-t)f(\theta_2)$。
- 而函数h_i为仿射变换（affine map），即存在a_i,b_i，使得$h_i(\theta) = \boldsymbol{a}_i \cdot \boldsymbol{\theta} + b_i$。可以理解为$h_i$为线性变换。
- 存在θ，对所有的函数g_i，都成立$g_i(\theta) < 0$。

[18] 定理的证明太过复杂，超出了本书的范围，有兴趣的读者请参考 Stephen Boyd 和 Lieven Vandenberghe 编写的 *Convex Optimization*。

书籍是人类进步的阶梯

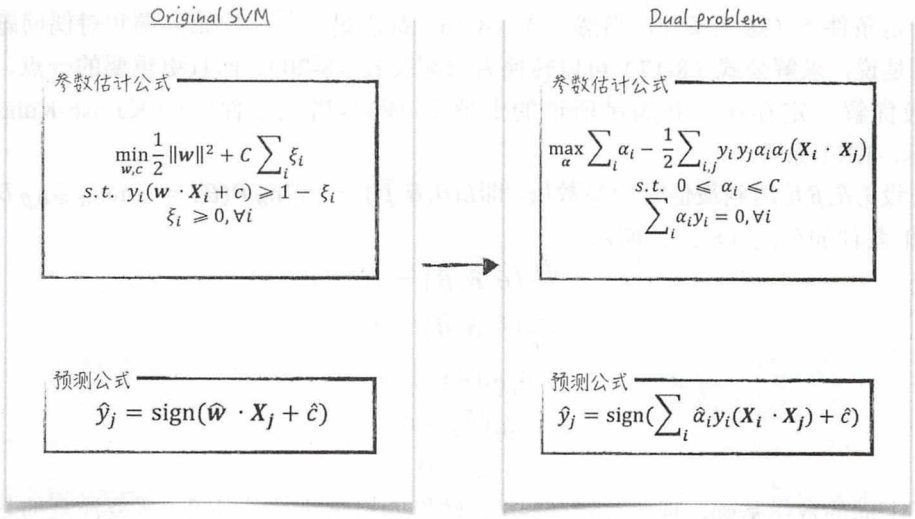


图 8-10

8.2.3 支持向量

将模型转换成相应的对偶问题能帮助我们更深刻地理解模型。比如公式（8-24）就清楚地展示了支持向量学习机是如何使用训练数据的，以及后者是如何影响预测结果的。

- 首先，分别计算被预测点X_j和训练数据里各点X_i的内积，即$\boldsymbol{X_i} \cdot \boldsymbol{X_j}$。
- 然后，将这些得到的内积作为权重去加权平均训练数据里的因变量$\hat{\alpha}_i y_i$，得到最终的预测值[19]。

事实上，由于向量内积$\boldsymbol{X_i} \cdot \boldsymbol{X_j}$在某种程度上可以度量$\boldsymbol{X_j}$和$\boldsymbol{X_i}$的相似度，因此支持向量学习机（几乎所有的算法模型也都这样）做模型的思路可以理解为：寻找与被预测数据相似的训练数据，并将相应的因变量加权平均得到最后的预测值。由此可以看到，模型在本质上是训练数据的一种数学组合（常常是线性组合），数据质量在很大程度上决定了最后效果的好坏。因此在机器学习领域有这样一个共识：数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。

回到公式（8-24），从表面上来看，每一个训练数据都会对预测结果产生影响，但实际上并非如此。不妨假设训练数据i满足$y_i(\hat{\boldsymbol{w}} \cdot \boldsymbol{X_i} + \hat{c}) > 1$，相应的约束条件为$g_i = 1 -$

[19] 事实上，很多模型的预测公式都能被写成这种向量内积的形式。比如之前提到的核岭回归，它的预测公式可表达为如下的形式，其中$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m) = (\boldsymbol{X}\boldsymbol{X}^{\mathrm{T}} + \lambda I)^{-1}\boldsymbol{Y}$，$m$为训练数据的个数。

$$\hat{y}_j = \sum_i \alpha_i (\boldsymbol{X_j} \cdot \boldsymbol{X_i})$$

书籍是人类进步的阶梯

$y_i(\hat{\boldsymbol{w}} \cdot \boldsymbol{X_i} + \hat{c}) - \hat{\xi}_i$。根据 KKT 条件，可得$\hat{\alpha}_i g_i = 0$，而$g_i = 1 - y_i(\hat{\boldsymbol{w}} \cdot \boldsymbol{X_i} + \hat{c}) - \hat{\xi}_i < 0$，这说明$\hat{\alpha}_i = 0$。

以二维空间为例，直观理解这一结论：对于如图 8-11 所示的模型结果，若数据落在两条虚线外，则它的权重肯定等于 0；只有落在虚线上或两条虚线内的点，它的权重才有可能不等于 0。这些点在学术上被称为支持向量（support vectors），这也是支持向量学习机这个名字的来源。

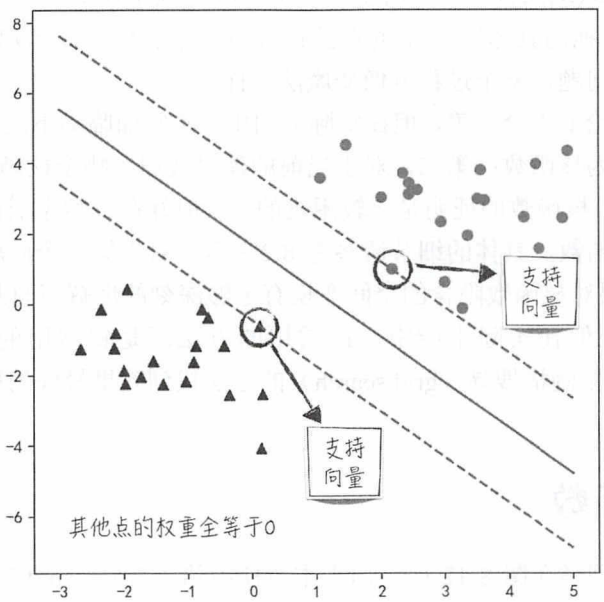


图 8-11

8.2.4 核函数的定义：优化运算

在研究了支持向量学习机的对偶问题后，接下来就将讨论什么是核函数以及如何使用它解决非线性分类问题。

与 8.2.1 节中的记号一致，不妨假设原始数据为$\{\boldsymbol{X_i}, y_i\}$以及非线性的空间变换为$\boldsymbol{\phi}$，那么支持向量学习机的实际训练数据是$\{\boldsymbol{\phi}(\boldsymbol{X_i}), y_i\}$。如果借助模型的对偶问题（事实上，只需将图 8-10 中的内积运算$\boldsymbol{X_i} \cdot \boldsymbol{X_j}$替换为$\boldsymbol{\phi}(\boldsymbol{X_i}) \cdot \boldsymbol{\phi}(\boldsymbol{X_j})$ 就可以得到空间变换后的最优化问题），完成模型训练其实并不需要$\boldsymbol{\phi}$的具体表达式，只需知道内积$\boldsymbol{\phi}(\boldsymbol{X_i}) \cdot \boldsymbol{\phi}(\boldsymbol{X_j})$。这就是所谓的核函数，记为$K(\boldsymbol{X_i}, \boldsymbol{X_j})$。

$$K(\boldsymbol{X_i}, \boldsymbol{X_j}) = \boldsymbol{\phi}(\boldsymbol{X_i}) \cdot \boldsymbol{\phi}(\boldsymbol{X_j}) \tag{8-26}$$

通过一个简单的例子来直观感受一下核函数。在 8.2.1 节中，我们通过坐标的平方来做

书籍是人类进步的阶梯

空间变换，现在将它稍稍复杂化一点。定义$\boldsymbol{\phi}(\boldsymbol{X}_i) = (x_{1,i}^2, x_{2,i}^2, \sqrt{2}x_{1,i}x_{2,i})$，则可以证明它的核函数如下：

$$\boldsymbol{\phi}(\boldsymbol{X}_i) \cdot \boldsymbol{\phi}(\boldsymbol{X}_j) = x_{1,i}^2 x_{1,j}^2 + x_{2,i}^2 x_{2,j}^2 + 2x_{1,i}x_{2,i}x_{1,j}x_{2,j}$$

$$\boldsymbol{\phi}(\boldsymbol{X}_i) \cdot \boldsymbol{\phi}(\boldsymbol{X}_j) = (\boldsymbol{X}_i \cdot \boldsymbol{X}_j)^2 = K(\boldsymbol{X}_i, \boldsymbol{X}_j) \tag{8-27}$$

对于一个已知的空间变换$\boldsymbol{\phi}$，使用核函数比直接计算向量内积$\boldsymbol{\phi}(\boldsymbol{X}_i) \cdot \boldsymbol{\phi}(\boldsymbol{X}_j)$更加高效。特别是当数据$\boldsymbol{\phi}(\boldsymbol{X}_i)$的维度很高时，核函数能极大地减少模型的运算量[20]。反过来，当在模型里使用某个核函数$K(\boldsymbol{X}_i, \boldsymbol{X}_j)$时，我们在不知不觉间完成了原始空间到某个新空间的映射（通常并不知道空间变换的具体形式），并在此过程中，将原本几乎无法解决的非线性问题转换为容易解决的线性问题，整个过程就像变魔法一样。

虽然核函数在理论上十分完美，但在实际应用中，常常面临如下的两大难题：第一，如何知道一个函数是否为核函数；第二，对于当前的建模数据，哪个核函数是最合适的。

对于第一个问题，核函数的证明是比较困难的[21]。但好在数学家们已经找到了一些对大多数场景都适用的核函数，具体的细节请参考 8.2.5 节。对于第二个问题，并没有一个特别完美的解决方法。如果对核函数隐含的空间变换有足够深刻的理解，可以通过分析问题场景，选择合适的核函数。但在实际生产中，最常用的方法还是将常用的核函数当成超参数（hyperparameter），使用网格搜寻（grid search）的方法找到效果最好的核函数，具体的做法请参考 4.3.3 节。

8.2.5 常用的核函数

常用的核函数被列举在图 8-12 中。由于本章使用第三方库 scikit-learn 搭建模型，为了便于读者使用开源算法，在介绍核函数时，将使用同算法 API 相同的参数记号。

借助 scikit-learn，在支持向量学习机中使用核函数非常简单。具体的代码如程序清单 8-2 所示。

（1）可以通过类 SVC 的“kernel”参数选择使用的核函数。既可以传入核函数名字符串，如 8.1.6 节中的“SVC(kernel='linear')”，也可以传入算法库里定义好的核函数类，如第 9～12 行代码所示。这两种方法的差别在于，前者支持的核函数选项较少，比如它不支持 Laplacian kernel，但后者支持。

---

[20] 以公式（8-27）所示的多项式变换为例，当$\boldsymbol{X}$的维度为d时，向量内积的计算复杂度为$O(d^2)$；而核函数的计算复杂度为$O(d)$。

[21] 核函数的证明通常使用 Mercer 定理。假设K为$\mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$的函数，若对于任意的$\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_m$，核矩阵（kernel matrix）$K_{i,j} = K(\boldsymbol{X}_i, \boldsymbol{X}_j)$为一个半正定矩阵（positive semi-definite），则函数K为一个核函数。

其中半正定矩阵的定义如下：矩阵$\boldsymbol{M}$称为半正定矩阵，当且仅当对于任意的非零向量$\boldsymbol{z}$，都有$\boldsymbol{z}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{z} \geqslant 0$。

书籍是人类进步的阶梯

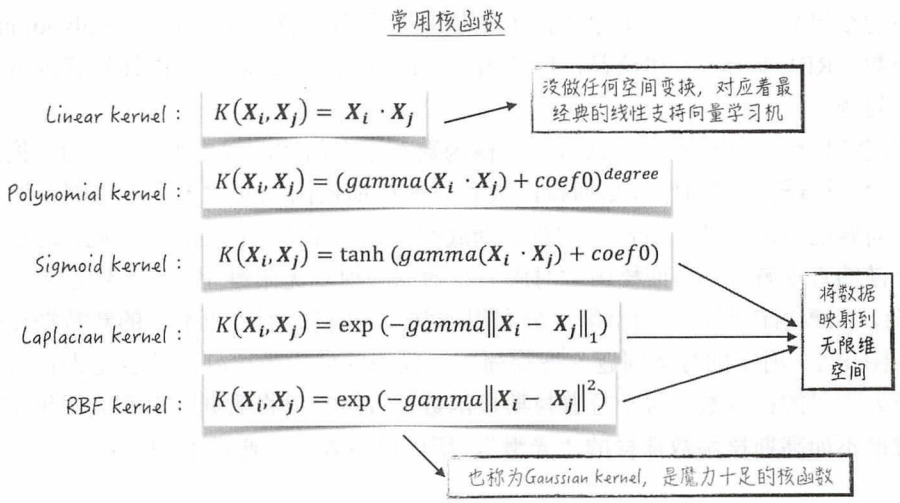


图 8-12

（2）对于核函数，SVC 还有一些其他的常用参数，比如“gamma”“coef0”以及“degree”等。它们对核函数的影响请参考图 8-12。正如 8.2.4 节中讨论的，这些参数其实都是模型的超参数，常使用网格搜寻的方法来找到它们最合适的取值。

程序清单 8-2 核函数[22]

```
from sklearn.svm import SVC
from sklearn.metrics.pairwise import linear_kernel, laplacian_kernel
from sklearn.metrics.pairwise import polynomial_kernel, rbf_kernel

def trainModel(data):
    """
    在模型里使用不同的核函数
    """
    kernel = [linear_kernel, polynomial_kernel, rbf_kernel, laplacian_kernel]
    res = []
    for i in kernel:
        model = SVC(kernel=i, coef0=1)
        model.fit(data[["x1", "x2"]], data["y"])
        res.append({"name": i.__name__, "result": model})
    return res
```

运行上面的代码，对图 8-9 里的“圆圈”数据做分类，可以得到如图 8-13 所示的结果。图中的圆点表示类别 1，三角形表示类别 0；灰色区域表示预测结果等于类别 0，而白色区域为预测结果等于类别 1。图中的实线是分离超平面，而虚线表示支持向量所在的范围。

[22] 完整的代码实现请参考随书配套的代码/ch08-supervised/svm/kernel.py。

书籍是人类进步的阶梯

模型的结果显示，线性支持向量学习机的效果很差，而多项式核函数（polynomial kernel），高斯核函数（RBF kernel）和拉普拉斯核函数（Laplacian kernel）都能比较好地解决这个非线性分类问题。

我们能够比较好地理解为什么多项式核函数对这个问题的效果很好：因为变换后的空间里恰好包含(x_1^2, x_2^2)。但为什么高斯核函数和拉普拉斯核函数的效果也不错呢？要很严谨地解释这个问题的原因需要比较深厚的数学功底[23]，超出了本书的范围。因此，这里只提供一个不太严谨的直观解释。高斯核函数对应着将数据映射到无限维空间，虽然这个空间并不包含(x_1^2, x_2^2)，但数据在变换后的空间里分布得非常“光滑”。而“光滑”的新数据使得很多问题（包括图 8-13 所示的分类问题）变得简单，模型的效果也很好。这也是为什么说高斯核函数是魔力十足的核函数。对于拉普拉斯核函数，虽然它也将数据映射到无限维空间，但变换后的数据不如高斯核函数那样的“光滑”，所以模型效果也通常不如后者。

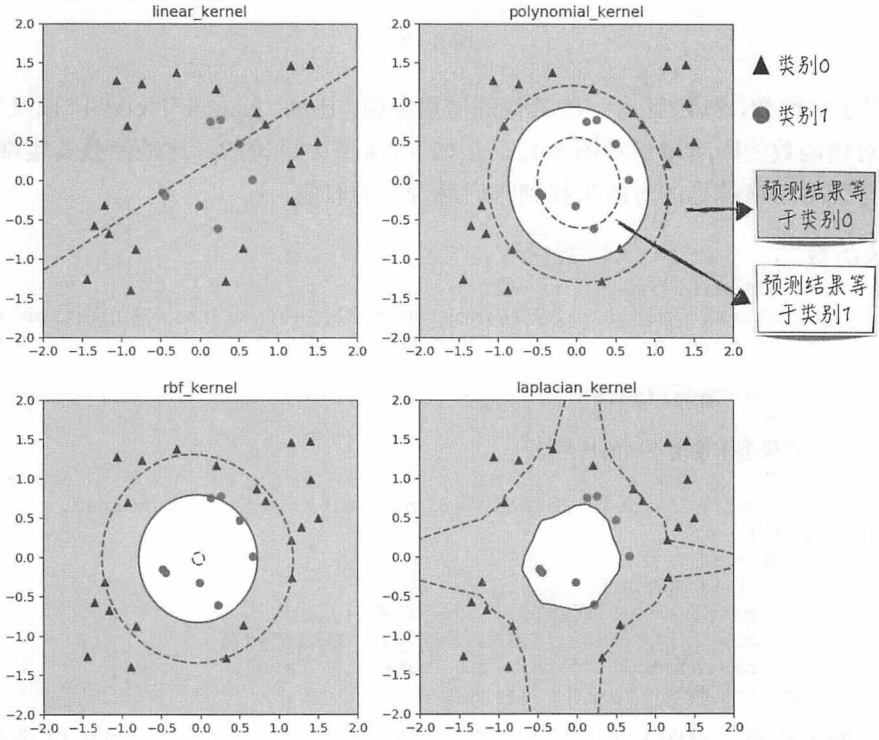


图 8-13

[23] 事实上，核函数对应着变换空间为函数空间（Reproducing Kernel Hilbert Spaces）。感兴趣的读者请参考 Herbrich R. 编写的 *Learning Kernel Classifiers. Theory and Algorithms*。

书籍是人类进步的阶梯

8.2.6 Scale variant

前面的章节中主要讨论了支持向量学习机的理论细节。本节作为介绍这个模型的最后一部分，将讨论在实际使用支持向量学习机时，常碰到但又经常被人忽略的一点：支持向量学习机与线性回归和逻辑回归不同[24]，它对特征（自变量）的线性变换不稳定。在学术上，这一点被称为 scale variant。

为了便于理解，先从一个具体的例子开始讲起。我们沿用 8.2.1 节中的非线性分类数据，这份数据的自变量有两个x_1, x_2。这两个变量的中心位置和变化幅度一样的，即它们的期望相同，方差也相等。使用支持向量学习机对数据建模，得到的预测结果来不错，如图 8-14 中的标记 1 所示。

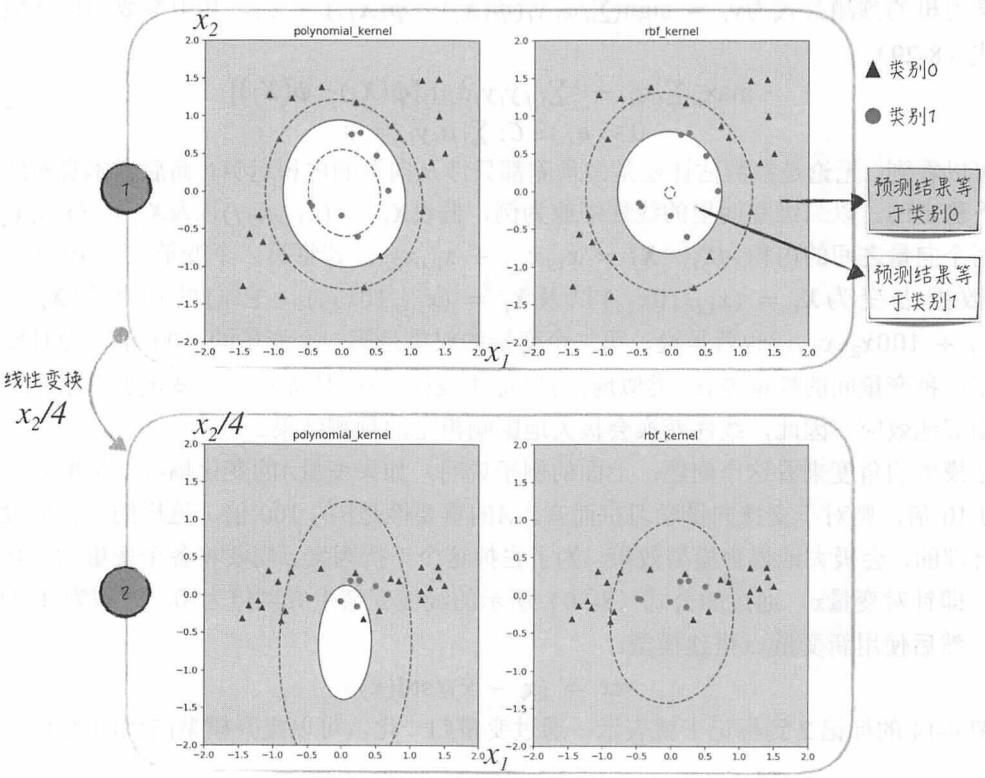


图 8-14[25]

[24] 更严谨的表述是：不带惩罚项的线性回归和逻辑回归模型对特征的线性变换是稳定的。

[25] 图 8-14 的代码实现请参考随书配套的代码/ch08-supervised/svm/scale_variant.py。

现在对数据的自变量做线性变换得到新的数据集，新数据的自变量为z_1, z_2，具体的定义如公式（8-28）所示。在实际生产中，类似的线性变换是经常发生的，比如更换变量x_2的计数单位。

$$z_1 = x_1, z_2 = x_2/4 \tag{8-28}$$

从直观上来讲，上面的变换并没有改变数据的相对位置，只是从整体上将数据“压扁”了一些，如图 8-14 中的标记 2 所示。按理来说，这个线性变换并不影响分类问题的解决[26]。但事实上，对于新的数据集，支持向量学习机的模型效果很差，比如使用高斯核函数时，所有数据的预测结果都是类别 0。

上面的例子说明，特征的线性变换（放大或缩小）将极大地影响模型的效果，也就是本节开头所提到的，支持向量学习机对特征的线性变换不稳定。这是因为从数学上来讲，支持向量学习机的预测公式为$\hat{y}_j = \text{sign}(\sum_i \hat{\alpha}_i y_i(\boldsymbol{\phi}(\boldsymbol{X}_i) \cdot \boldsymbol{\phi}(\boldsymbol{X}_j)) + \hat{c})$，其中参数的估计值依赖于公式（8-29）。

$$\begin{gathered}\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} y_i y_j \alpha_i \alpha_j [\boldsymbol{\phi}(\boldsymbol{X}_i) \cdot \boldsymbol{\phi}(\boldsymbol{X}_j)] \\ 0 \leqslant \alpha_i \leqslant C;\ \sum_i \alpha_i y_i = 0\end{gathered} \tag{8-29}$$

可以看到，无论是参数估计还是做预测都只涉及向量的内积运算，而后者本身对线性变换是不稳定的。以二维空间里的线性缩放为例，假设$\boldsymbol{X}_i = (x_{1,i}, x_{2,i})$以及$\boldsymbol{X}_j = (x_{1,j}, x_{2,j})$，则这两个向量之间的内积为$\boldsymbol{X}_i \cdot \boldsymbol{X}_j = x_{1,i}x_{1,j} + x_{2,i}x_{2,j}$。若将第二个变量扩大 10 倍，则相应的数据点变为$\boldsymbol{X}_i = (x_{1,i}, 10x_{2,i})$以及$\boldsymbol{X}_j = (x_{1,j}, 10x_{2,j})$，它们的内积为$\boldsymbol{X}_i \cdot \boldsymbol{X}_j = x_{1,i}x_{1,j} + 100x_{2,i}x_{2,j}$。也就是说，第二个变量的权重是第一个变量的 100 倍，而且模型无法修正这种变量间的权重差异。类似地，可以证明线性平移，比如第二个变量加上某个常数，也有相似地效应。因此，线性变换会极大地影响模型的预测结果。

从模型的角度来看这个问题，上面的例子说明，如果变量A的变化幅度（标准差）是变量B的 10 倍，则对于支持向量学习机而言，A的重要性是B的 100 倍。这样的隐含假设显然是不合理的，会极大地损害模型效果。为了去掉这个干扰因素，需要将各个变量归一化后使用[27]。即针对变量x，通过如公式（8-30）所示的线性变换生成期望为 0，方差为 1 的新变量xx，然后使用新变量xx搭建模型。

$$xx = (x - \bar{x})/\text{std}(x) \tag{8-30}$$

图 8-14 的标记 2 到标记 1 就表示：通过变量归一化，可以提升模型的预测效果。

---

[26] 若将坐标系的轴坐标步长缩短为现在的 1/4，则新数据的图像和原数据的图像将一模一样。

[27] 另一种常用的方法是将变量x映射到某个特定的区间，比如$[-1, 1]$。相应的变换公式为：

$$xx = 2 \cdot \frac{x - \min(x)}{\max(x) - \min(x)} - 1$$

8.3 决策树

决策树（decision tree）是一个十分有趣的模型，它的建模思路是尽量模拟人做决策的过程。因此，决策树与其他大多数机器学习模型不同，它几乎没有任何数学抽象，完全通过生成决策规则来解决分类和回归问题。

这个模型在学术上被归为白盒模型（white box model），因为它的整个运行机制能很直接地被翻译成人类语言，即使对建模这个领域完全不了解的非技术人员也能很好地理解它。决策树模型的核心是决策规则。因此在讨论具体的模型细节之前，我们先来看看在现实生活中，人们是如何利用规则做决定的。

8.3.1 决策规则

在周五的晚上，数据科学家小安在考虑这个周末她要做什么。

- 首先，小安手上并没有未完成的工作，不需要周末加班，因此她选择周末休息。
- 然后，小安上网查看了一下最近上映的电影，里面并没有她感兴趣的。因此她决定不去电影院，而选择其他的娱乐方式。
- 最近，小安查看了天气预报，这周末的天气非常炎热，气温高达 40 度。因此她打消了出去逛街的念头，最终决定待在家里打游戏。

小安做决策的整个过程如图 8-15 所示，其中圆角框表示决策规则，方形框表示决策结果，框与框之间的连线表示决策路径。

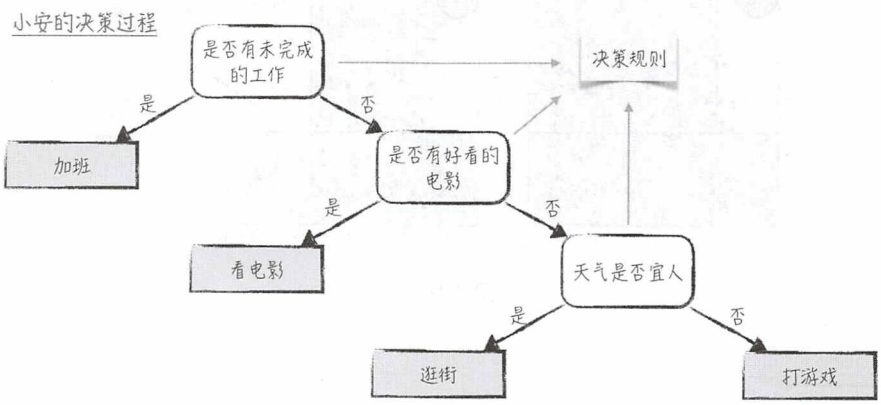


图 8-15

决策树[28]想要搭建的就是同图 8-15 类似的树形决策流程图。为了更直观地理解模型，我们来看一个具体的例子。随机生成如图 8-16 所示的数据（图形符号的含义和之前的章节一样），数据有两个自变量x_1, x_2，决策树按如下的步骤将数据分为两类。

（1）首先，根据规则$x_2 \leqslant -0.2994$将平面分为两块区域，如图 8-16 中标记 1 所示。由于下面的灰色区域内，所有训练数据都是类别 0，因此，这个区域的预测结果为类别 0。同理，上面的白色区域内，类别 1 的占比更高，因此，这个区域的预测结果为类别 1。

（2）接着，在第 1 步的基础上，新生成规则$x_2 \leqslant -0.6147$，并以此将平面细分为 3 个区域，如图 8-16 中标记 2 所示。同第 1 步类似，根据区域内各类别的占比，确定该区域的预测结果。

（3）重复上面的过程，可以得到如图 8-16 中标记 3 所示的结果。事实上，决策树不但可以预测数据的类别，还可以得到数据属于各个类别的概率。比如标记 3 中的右边白色区域，它里面有 10 个数据点是类别 0，4 个数据点是类别 1，因此，这个区域属于类别 1 的概率为$10/(10+4)$，属于类别 0 的概率为$4/(10+4)$。

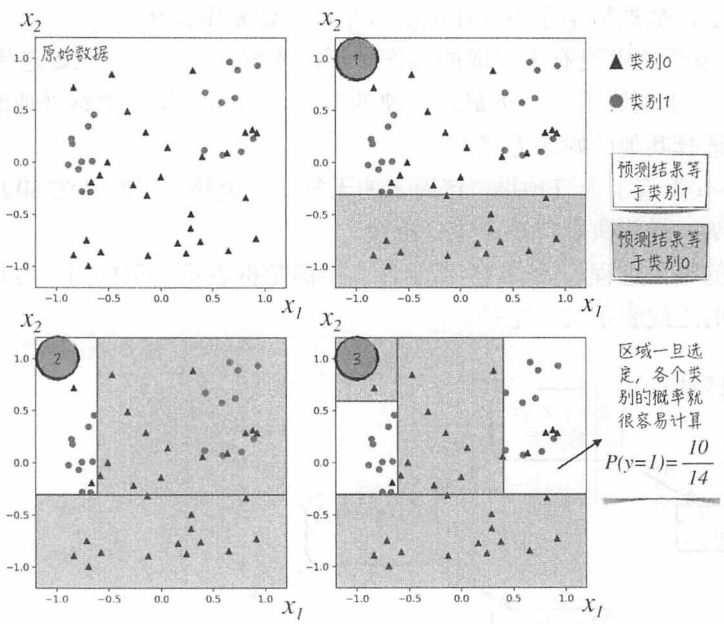


图 8-16[29]

[28] 决策树模型其实有很多种不同的版本，而本章所讨论的只是决策树模型中应用最广的一种。该模型在学术上被称为 CART（Classification And Regression Trees），它通过构造二叉树来解决分类和回归问题。

[29] 图 8-16 的代码实现请参考随书配套的代码/ch08-supervised/decision_tree/simple_example.py。

可以将上面的模型结果转换为更为直观的树形结构，如图 8-17 所示。图中的方框表示树的一个节点，其中带有决策规则的节点被称为中间节点（internal node），没有决策规则的为叶子节点（leaf node）。叶子节点表示预测结果，中间节点表示决策过程，而节点与节点之间的连线表示决策路径。方框中各个字段的含义已在图中给出，其中“gini”字段表示节点的不纯度，具体的细节将在 8.3.2 节中讨论。

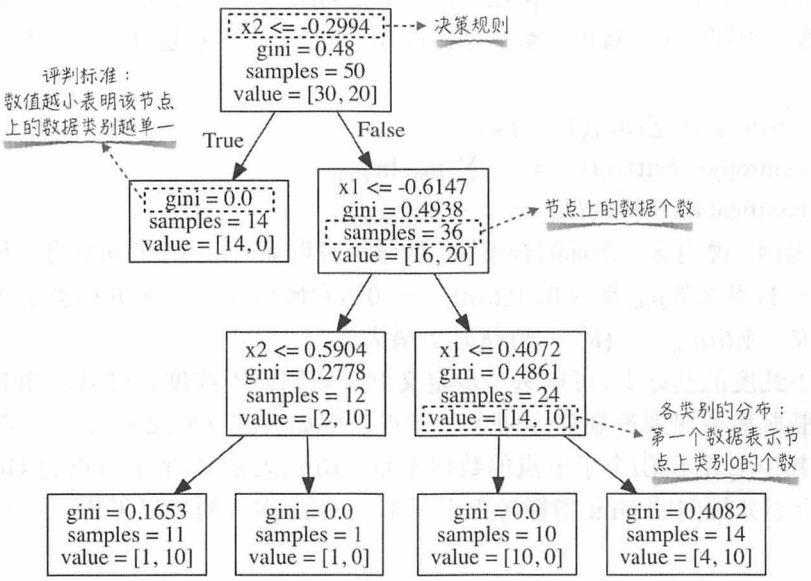


图 8-17

由上面这个简单的例子可以看到，决策树的规则生成方法为：选择一个划分变量和相应的阈值，变量取值大于阈值为一种决策路径，否则为另一种决策路径[30]，因此，决策树的决策规则也称为划分规则。下面将讨论划分规则（决策规则）的评判标准，这是决策树选择最优划分变量和相应阈值的基础。

8.3.2 评判标准

针对分类问题，划分规则的评判可以分为两步。

- 如果一个节点上的数据都差不多是同一个类别，那么，这个节点就几乎不需要再做划分了，否则需要针对该节点，生成新的划分规则。

[30] 严格来说，这只是针对数值型变量的划分原则。针对类别型变量，需要先将其转换为多个虚拟变量，再使用同样的原则对虚拟变量做划分。

仅供非商业用途或交流学习使用

- 如果新的规则能基本上把节点上不同类别的数据分离开，使得每个子节点上都是类别比较单一的数据，那么这个规则就是一个好规则。

将上面的两个原则翻译成数学语言。假设数据分为K类，分别记为$0, 1, \cdots, K-1$；当前节点记为m，节点上一共有N_m个数据。定义类别i在该节点上的占比如下：

$$p_{mi} = \left(1/N_m\right) \sum_j 1_{\{y_j = i\}} \tag{8-31}$$

现在定义该节点的不纯度（impurity），通常记为H_m。该指标的取值区间为[0, 1]，用于衡量节点上数据类别的单一程度，数值越接近 0，表示数据类别越单一。常用的指标有如下几种。

- Gini：$Gini_m = \sum_i p_{mi}(1 - p_{mi})$
- Cross entropy：$Entropy_m = -\sum_i p_{mi} \ln p_{mi}$
- Misclassification：$Misclf_m = 1 - \max_i p_{mi}$

以 Gini 为例，说明这些指标的合理性。数学上可以证明，当节点m只有一种类别时，不妨设为$p_{m0} = 1$，其余的p_{mi}都为 0，则$Gini_m = 0$达到最小值。当节点m的类别均匀分布时，即$p_{mi} = 1/K$，则$Gini_m = (K-1)/K$达到最大值。

在节点不纯度的基础上，可以进一步定义划分规则的不纯度。依然以 Gini 指标为例，假设节点m根据某种规则被划分为两个子节点，则如公式（8-32）所示定义划分规则的 Gini 指标，其中N_i表示第i个子节点的数据个数，$Gini_i$表示第i个子节点的 Gini 指标。也就是说，某个划分规则的 Gini 指标等于个子节点 Gini 指标的加权平均，权重为子节点的数据量占比。

$$Gini_{split} = \sum_{i=1}^{2} \left(N_i/N_m\right) Gini_i \tag{8-32}$$

接下来，我们讨论划分规则的处理方法。

- 当节点m的 Gini 指标小于等于某个阈值（不妨记为 min_impurity_split）时，则表示该节点不需要进一步拆分，否则需要生成新的划分规则（当然，节点m停止划分的条件还有其他一些，具体的细节将在 8.3.5 节中讨论）。
- 对于每一个需要再次划分的节点，选择 Gini 指标最低的划分规则来生成子节点，并不断重复这个过程，直至所有节点都不需要再次划分。如果读者对算法比较了解，会发现决策树的划分策略其实就是贪心算法[31]。

上面的讨论针对的是分类问题，其实决策树也能解决回归问题，具体过程和分类问题大同小异，唯一的区别就是将不纯度的评判标准改为距离误差，比如均方差（Mean Squared Error，MSE），具体公式请参考 4.1.1 节。

[31] 寻找最优的决策树是一个 NP 完全问题，因此只能退而求其次地使用贪心算法求解。

仅供非商业用途或交流学习使用

8.3.3 代码实现

针对一个分类问题或回归问题，根据上面讨论的评判标准和划分规则的算法就可以生成相应的决策树，其代码实现也很简单，如程序清单 8-3 所示。

（1）针对分类问题，DecisionTreeClassifier 是第三方库 scikit-learn 提供的实现方法，如第 1 行代码所示。如果需要使用决策树解决回归问题，则需要使用 sklearn.tree 下面的另一个类 DecisionTreeRegressor。

（2）可以通过 DecisionTreeClassifier 里的参数“criterion”，选择不同的不纯度指标，如第 9 行代码所示。截至 0.18 版本，scikit-learn 支持的选项为“gini”和“entropy”（对应着 8.3.2 节中的 Cross entropy）。

（3）DecisionTreeClassifier 里有一些控制决策树复杂度的参数，比如第 9 行代码中的“max_depth”。它表示决策树的最大高度，当树的高度（比如图 8-17 中，决策树的高度为 3）达到这个参数时，即使树中仍有不纯度较高的节点，决策树也会停止继续划分。常用的类似参数还有“max_leaf_nodes”表示决策树的最大叶子节点数，这个参数被设置之后，模型会使用最佳优先搜寻算法（best-first search）来保证树的叶子节点不大于这个参数；“min_samples_split”表示当节点的数据量大于这个参数时，决策树才会继续划分节点；“min_samples_leaf”表示当叶子节点的数据量大于这个参数时，相应的叶子节点才会被保留，否则会被剪枝；“min_impurity_split”表示当节点的不纯度大于这个值时，决策树才会继续划分节点。

程序清单 8-3 决策树

```
1  from sklearn.tree import DecisionTreeClassifier
2
3  def trainModel(data):
4      """
5      训练决策树模型
6      """
7      res = []
8      for i in range(1, 4):
9          model = DecisionTreeClassifier(criterion="gini", max_depth=i)
10         model.fit(data[["x1", "x2"]], data["y"])
11         res.append(model)
12     return res
```

8.3.4 决策树预测算法以及模型的联结

正如前面提到的，决策树是所谓的白盒模型，它预测的算法很容易解释：对于一个给定的数据，根据决策树里各个节点的决策规则，将数据分配到某个叶子节点上，并由叶子节点给出最终的预测值，整个过程如图 8-18 所示。

仅供非商业用途或交流学习使用

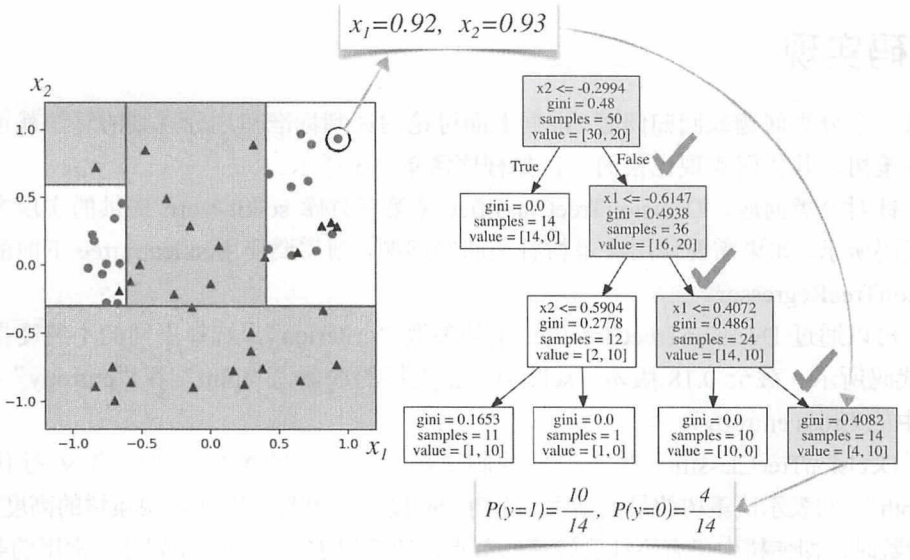


图 8-18

在决策树模型中，叶子节点提供预测结果的算法十分简单。

- 针对分类问题，叶子节点给出的其实是数据属于每个类别的概率，而这个概率值等于各个类别的数据占比。严谨一点，假设数据分为K类，分别记为$0, 1, \cdots, K-1$，叶子节点上一共有N个数据，则$P(y=i)=\sum_j 1_{\{y_j=i\}}/N$。基于预测得到的概率，就很可以很直接地得到最终的预测结果为出现概率最大的类别。
- 针对回归问题，叶子节点的处理方式类似，最终的预测结果等于节点内标签变量$\{y_i\}$的平均值。

仔细分析决策树可以得到，这个模型的优点在于能综合考虑多个变量，而且对变量的线性变换是稳定的。另外，它对连续性变量的处理方法是将其划分成几个互不相交的区间，这样的处理方法能有效地规避定量变量边际效应恒定的隐含假设(具体的细节请参考 7.3 节)。但模型的缺点也同样明显，模型的最后一步算法比较薄弱，只是简单地求类别占比或平均值。这导致单独使用决策树搭建模型时，预测效果并不理想（在实际建模中，我们很少单独使用决策树模型）。

在实际应用中，为了得到更好的预测效果，需要借助 7.6.5 节中曾讨论过的模型联结主义（connectionism）：将决策树作为整体模型的一部分和其他模型嵌套起来使用。模型联结的方式有很多，并没有一个通用的解决方案。根据问题场景，设计合适的模型架构，正是数据科学家发挥自己创造力和想象力的舞台。

这里仅当抛砖引玉，简单讨论一个在广告行业（RTB）和金融反欺诈领域里应用得很广

仅供非商业用途或交流学习使用

的模型联结方式，如图 8-19 所示。具体来说，我们可以将决策树视为一种特征提取的模型，首先使用它对某些原始特征（一般为数值型特征）做聚类运算，将数据位于决策树的哪个叶子节点作为新特征。比如假设决策树有 4 个叶子节点，依次命名为 1、2、3、4；某个数据落在了第 3 个叶子节点，则用向量(0, 0, 1, 0)（这就是上面提到的新特征）来表示这个数据。然后利用这些新特征和剩下的原始特征搭建逻辑回归模型，并由此得到最终的预测结果[32]。

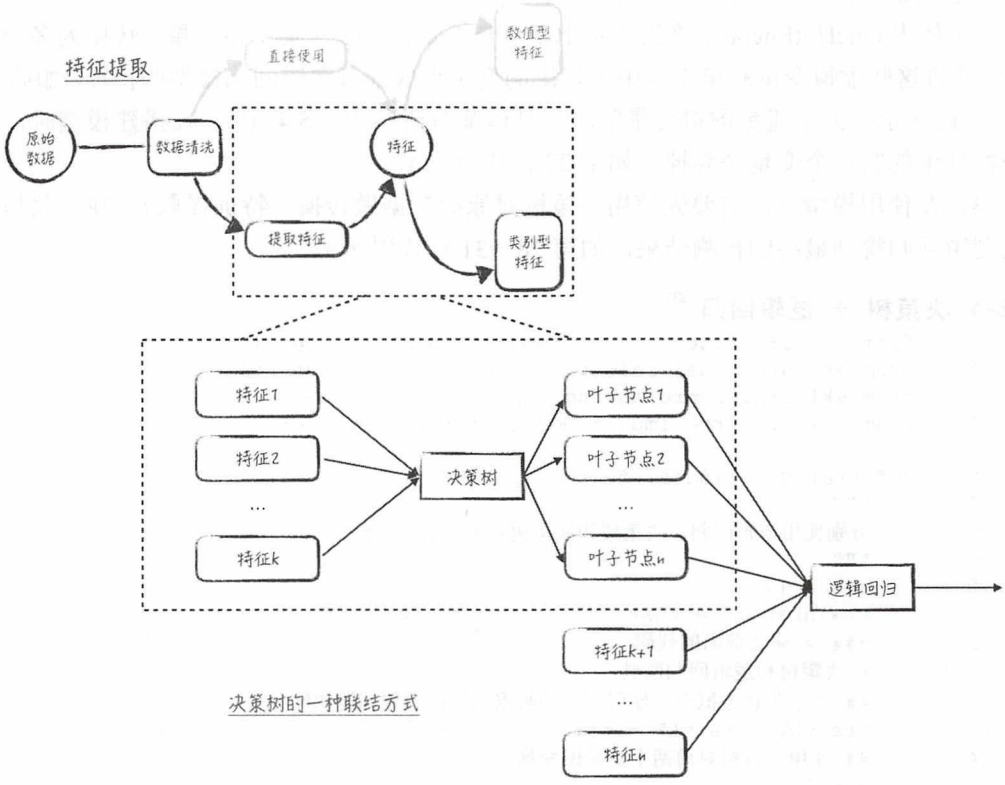


图 8-19

如果使用这样的模型架构，那么决策树的最后一步算法其实并没有被触发。这样，模型就只保留了决策树模型的优点，并且在此基础上又引入了逻辑回归预测效果好，稳定性高的长处。

值得注意的是，如果在图 8-19 所示的模型中，特征1到特征k都是数值型变量，那么决策树的作用其实是将数值型变量（定量变量）转换为了类别型变量（定性变量）。这其实是

[32] 在实际生产中，为了提升模型效果，通常会将决策树替换为效果更好但也更为复杂的树形模型 gradient-boosted trees（GBTs），但决策树和 GBTs 在整个模型中的作用是完全一样的。

仅供非商业用途或交流学习使用

7.3 节所讨论内容的一种延伸：根据卡方检验（chi-squared test），每次只能将一个数值型变量转换为类别型变量，因此变量之间的联动效应在转换过程中就被忽略了。但是使用决策树，则可以综合地考虑多个数值型变量。

程序清单 8-4 展示了如何用 Python 代码实现如图 8-19 所示的模型。

（1）对于训练好的决策树模型“_dt”，可以函数_dt.apply 得到新变量“leafNode”。这个特征表示数据所在的叶子节点，如第 20 行代码所示。

（2）使用 OneHotEncoder 将第 1 步中新变量“leafNode”（类别型变量）转换为多个虚拟变量；并将这些虚拟变量和第 1 步中未使用的变量组成训练逻辑回归模型的特征，如第 21～27 行代码所示。为了避免虚拟变量陷阱（具体细节请参考 7.5.4 节），在搭建模型时，需要将新特征中的第一个变量舍弃掉，如第 27 行代码所示。

（3）在使用模型时，需要先使用决策树对原始数据做转换（特征提取），并在此基础上使用逻辑回归得到最终的预测结果，如第 28～31 行代码所示。

程序清单 8-4 决策树 + 逻辑回归[33]

```
1  | from sklearn.linear_model import LogisticRegression
2  | from sklearn.model_selection import train_test_split
3  | from sklearn.preprocessing import OneHotEncoder
4  | from sklearn.tree import DecisionTreeClassifier
5  |
6  | def trainModel(data, features, label):
7  |     """
8  |     分别使用逻辑回归、决策树和决策树+逻辑回归建模
9  |     """
10 |     res = {}
11 |     trainData, testData = train_test_split(data, test_size=0.5)
12 |     ### # 略去前面的代码
13 |     # 决策树和逻辑回归联结
14 |     ## 为了防止过拟合，使用不同的数据训练决策树和逻辑回归
15 |     trainDT, trainLR = train_test_split(trainData, test_size=0.5)
16 |     ## 使用决策树对前两个变量做变换
17 |     m = 2
18 |     _dt = DecisionTreeClassifier(max_depth=2)
19 |     _dt.fit(trainDT[features[:m]], trainDT[label])
20 |     leafNode = _dt.apply(trainDT[features[:m]]).reshape(-1, 1)
21 |     coder = OneHotEncoder()
22 |     coder.fit(leafNode)
23 |     newFeature = np.c_[
24 |         coder.transform(_dt.apply(trainLR[features[:m]]).reshape(-1, 1)).toarray(),
25 |         trainLR[features[m:]]]
26 |     _logit = LogisticRegression()
27 |     _logit.fit(newFeature[:, 1:], trainLR[label])
28 |     testFeature = np.c_[
```

[33] 完整的实现请参考随书配套的代码/ch08-supervised/decision_tree/decision_tree_logit.py。

仅供非商业用途或交流学习使用

```
        coder.transform(_dt.apply(testData[features[:m]]).reshape(-1, 1)).toarray(),
        testData[features[m:]]]
    dtLogitProb = _logit.predict_proba(testFeature[:, 1:])[:, 1]
    res["DT + logit"] = roc_curve(testData[label], dtLogitProb)
    return res
```

为了验证模型效果，我们模拟一份二元分类问题的数据，并对数据分别搭建逻辑回归、决策树以及决策树加逻辑回归模型。可以得到如图 8-20 所示的结果，图中 logit 表示逻辑回归、DT 表示决策树、DT + logit 表示决策树加逻辑回归。

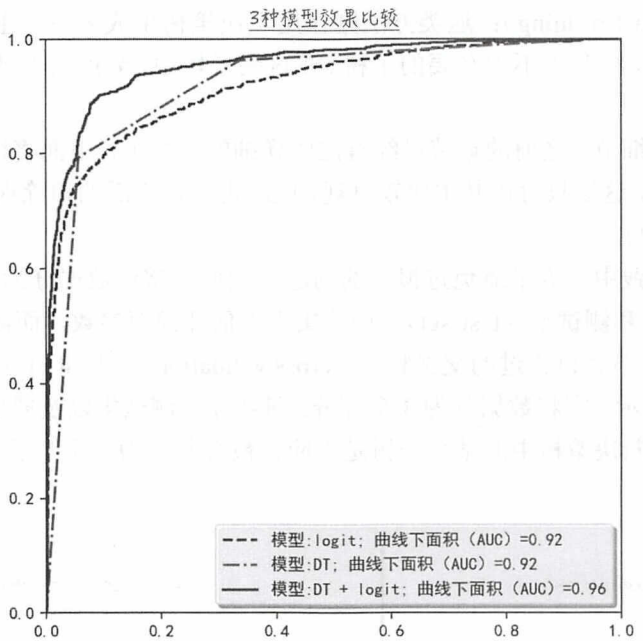


图 8-20

8.3.5 剪枝

在实际应用中，决策树模型常常会遇到过拟合的问题[34]。从直观上理解，决策树几乎可以无限制地划分树中的节点，使得每个叶子节点里只剩下极少量的、类别较为单一的数据。这会导致表面上，模型的预测结果很完美（针对训练数据），但实际上，模型完全没有捕捉

[34] 从理论的角度上来讲，决策树属于非参模型（nonparametric model）。也就是说，决策树隐含的模型参数个数和训练数据一样多（严格来讲，假设训练数据为n，而模型的参数个数为$O(n)$，则模型为非参模型。因此非参模型并不是没有参数，而是指有“无穷多”个参数）。因此，相比于参数个数固定的参数模型（parametric model），比如逻辑回归，决策树更容易引发过拟合的问题。

仅供非商业用途或交流学习使用

到数据间的内在联系。

为了解决这个问题，需要对症下药：用一些技术手段防止决策树过细地划分节点，这在学术上被形象地称为剪枝（pruning）。根据使用时间点的不同，具体的方法又可以细分为以下两类。

- 前剪枝（pre-pruning）：这类方法作用于决策树的生成过程中，通过一些阈值来限制决策树的生长，比如之前章节讨论过的 max_depth、min_samples_split 等参数。这种方法在实际中使用得较多。
- 后剪枝（post-pruning）：这类方法则作用于决策树生成之后，主要的思路是对于一棵已生成的决策树，将其中不太必要的子树（在这些子树中，节点的不纯度下降得不明显）剪掉。

关于前剪枝的细节，之前的章节已经有比较详细的介绍了，下面将讨论后剪枝。具体的后剪枝算法有不少，这里只讨论其中比较直观，应用也比较广泛的消除误差剪枝法（Reduced Error Pruning，REP）。

在搭建模型实践中，为了避免过拟合的问题，我们常常将数据分为互不相交的两部分：训练集（train set）和测试集（test set）。训练集用于估计模型参数，而测试集用于评估模型效果。这就是 4.3.1 节曾讨论过的交叉验证（cross validation）[35]。REP 是这个方法的进一步延伸，如图 8-21 所示，它将数据分为 3 个部分：训练集、测试集以及剪枝集（pruning set）。其中剪枝集用于判断决策树中的某个子树是否应该被合并成为一个叶子节点，也就是所谓的剪枝。

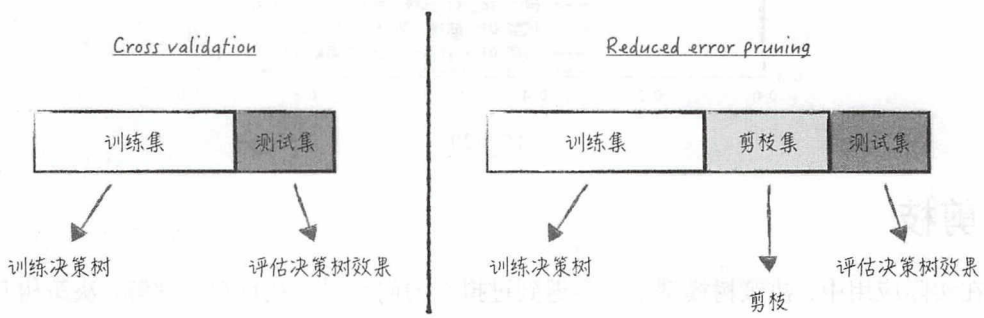


图 8-21

现在用一个简单的例子来说明 REP 算法的具体步骤。搭建模型的数据分为 A、B 两个类别，使用的自变量有两个，记为x_1, x_2。我们使用训练数据训练模型，得到了如图 8-22 中标

[35] 在实际中，更严谨的交叉验证是将数据分为 3 份：训练集、验证集和测试集。在这种情况下，REP 的做法是将数据分为 4 份，在交叉验证的基础上增加一个剪枝集。

仅供非商业用途或交流学习使用

记 1 所示的决策树。与之前的记号类似，图中的方框表示树的节点；方框中的第一行表示节点的 ID，比如 v1，v2 等；方框的第二行表示该节点里各个类别的数据量（针对训练数据），比如对于节点 v4，values=[1, 6]表示类别 A 的数据只有 1 个，而类别 B 的数据有 6 个，那么根据决策树的预测算法，节点 v4 的预测结果为类别 B。图中箭头上的公式表示决策规则，比如当$x = 0$时，数据会从节点 v1 移动到节点 v2。

使用如图 8-22 中标记 2 所示的剪枝集去修正已训练好的决策树。

- 对于 D1，它在节点 v4，那么根据模型，预测结果为类别 B 是错误的。同理，对于 D2，它的预测结果也是错误的。如果将这两个节点剪掉，那么 D1 和 D2 都落在节点 v2 上，这时只有 D2 的预测结果是错误的。也就是说，针对子数 v2、v4、v5，在剪枝之前预测结果错误的个数是两个，而剪枝之后，预测结果错误的个数下降为一个。这种情况下，REP 就会执行剪枝操作。

- 类似地，用数据 D3、D4、D5 对子树 v3、v6、v7 做评估，可以得到剪枝之前的错误数为 1 个（D4），剪枝之后的错误数也为 1 个（D5），这种情况下，REP 就不会执行剪枝操作。

- 由于 v3 的子树被没有被剪掉，因此不考虑是否对子树 v1、v2、v3 进行剪枝，这在学术上被称为 bottom-up restriction。

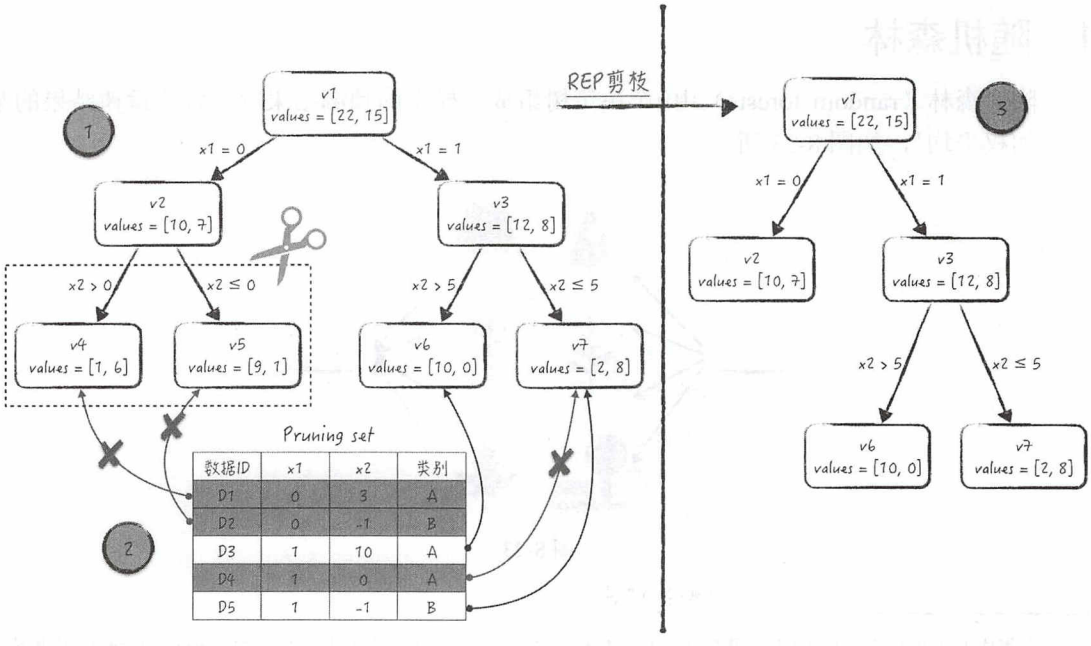


图 8-22

剪枝完成后，会得到如图 8-22 中标记 3 所示的决策树，REP 的剪枝步骤可总结如下。

- 根据剪枝集，定义节点v的剪枝收益，假设以v为根节点的子树为T。

$$Gain_v = T\text{的预测错误数} - v\text{的预测错误数} \tag{8-33}$$

- 从叶子节点开始，往上剪去所有收益大于 0 的子树，除非子树中存在剪枝收益为负的节点。

8.4 树的集成

决策树模型虽然简单明了，但单独使用时效果并不理想。因此，8.3.4 节讨论了决策树和逻辑回归配合使用方法。将两种模型联结成为一个整体后，模型的效果得到了明显的提升。但由于其中涉及两种完全不同的模型，在数学上很难给这种联结方式一个比较理想的抽象，因此在工程实现上，很难做到自动寻找最优的模型组合。

为了使模型间的组合更加自动化，最常规或者最成熟的做法就是只使用一种模型，比如决策树。通过某种方式将多个决策树组合起来，使用它们的“集体智慧”来解决问题，学术上被称为集成方法（ensemble method）[36]。

针对决策树的集成方法通常可以被分为两类：平均方法（averaging methods）和提升方法（boosting methods）。它们的代表模型分别是随机森林和 GBTs。

8.4.1 随机森林

随机森林（random forests）由n个决策树组成，模型的预测结果等于各决策树结果的某种“加权平均”，如图 8-23 所示。

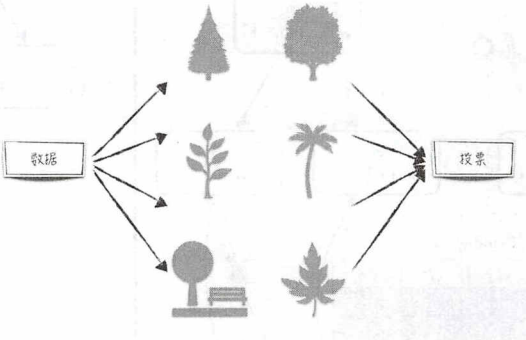


图 8-23

[36] 集成方法并不仅限于决策树。事实上，可以把机器学习中比较简单的模型称为弱学习（weak learner），而集成方法指的是将一系列弱学习组合成一个预测效果更好的复杂模型。这种方法与之前讨论的联结主义比较类似，区别在于联结主义更侧重于哲学层面，而集成方法则更侧重于具体算法。

- 对于分类问题，最终结果等于在决策树预测结果中出现次数最多的类别。直观上，可以将每个决策树想象成一个人，而随机森林想象成一场投票，通过少数服从多数的原则得到最终的结果。
- 对于回归问题，最终结果等于决策树预测结果的平均值。

随机森林的建模依据是：一棵树犯错的概率比较大，但很多树同时犯错的概率就很小了。通过一个简单的例子，从数学上来证明一下这种做法的正确性。假设针对某个分类问题，有3棵相互独立的决策树，它们各自预测错误的概率为20%。如果将它们按少数服从多数的原则组合起来，形成一个随机森林，那么预测犯错的情况可分为如下两种：3棵决策树都错误或者只有一棵树预测正确。计算可得这个随机森林的犯错概率下降到10.4%。

$$0.2^3 + 3 \times 0.2^2 \times 0.8 = 10.4\% \tag{8-34}$$

由上面的例子可以看到，随机森林预测效果最重要的保证是森林中的决策树是相互独立的（极端地，假设森林中的每棵树都是一样的，则随机森林模型等同于决策树模型）。那么针对同一份训练数据，应该如何产生随机的决策树呢？

首先回顾一下决策树划分节点的具体步骤。使用训练集中的所有数据（假设数据被分为训练集和测试集）训练模型。对于需要划分的节点，选择最优的一个自变量以及相应的阈值，将其划分为左右两个子节点，使得子节点的不纯度之和达到最小（具体细节请参考8.3.1节和8.3.2节）。因此，可以从如下的3个层面引入决策树的随机性。

- 对于每个决策树，从原始训练集中随机选取训练该决策树的数据。
- 在划分节点时，并不遍历全部自变量，而是随机挑选其中的一部分作为候选自变量。
- 在选择自变量的划分阈值时，并不求得最优的解，而是随机构成一个候选阈值集合，并从中选取效果最优的（子节点的不纯度之和最低）。

针对随机森林，第三方库 scikit-learn 提供了两种实现：random forests 和 extremely randomized trees。这两者的差异在于 random forests 实现了随机性中的前两点，即训练数据随机以及候选自变量随机，而 extremely randomized trees 实现了随机性中的全部3点。在代码层面，这两类模型被封装成4个类（模型的参数和调用代码与决策树类似，在此不再赘述，有兴趣的读者请参考 scikit-learn 官方网站）。

- random forests 包括两个类：用于分类问题的 RandomForestClassifier 和用于回归问题的 RandomForestRegressor。
- extremely randomized trees 也包含两个类：ExtraTreesClassifier 和 ExtraTreesRegressor。它们分别解决分类问题和回归问题。

8.4.2 Random forest embedding

随机森林虽然是监督式学习，但它其实同样能处理没有标签变量的数据（或者故意不使

用数据中的标签变量）。也就是说随机森林能被当作非监督式学习的模型使用[37]，先来看看这种方法的具体步骤。

假设训练数据一共有n个且数据由m维向量表示，记为$\{\boldsymbol{X}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,m}), 1 \leqslant i \leqslant n\}$。将这$n$个原始数据都归为一类，也就是说凭空生成标签变量$\{y_i = 0, 1 \leqslant i \leqslant n\}$。然后按如下的方法生成合成数据$\{\boldsymbol{X}_i, n+1 \leqslant i \leqslant 2n\}$：对于数据$\boldsymbol{X}_{n+1}$，从原始数据第 1 个变量的取值里（即$\{x_{i,1}\}$）随机抽取一个作为它的第 1 个变量；从原始数据第 2 个变量的取值里（即$\{x_{i,2}\}$）随机抽取一个作为它的第 2 个变量，并以此类推，生成合成数据$\boldsymbol{X}_{n+1}$。不断重复上面的这个过程，直到生成n个新的合成数据。数据生成完成之后，将合成数据归为另一类，即$\{y_i = 1, n+1 \leqslant i \leqslant 2n\}$，整个过程如图 8-24 所示。

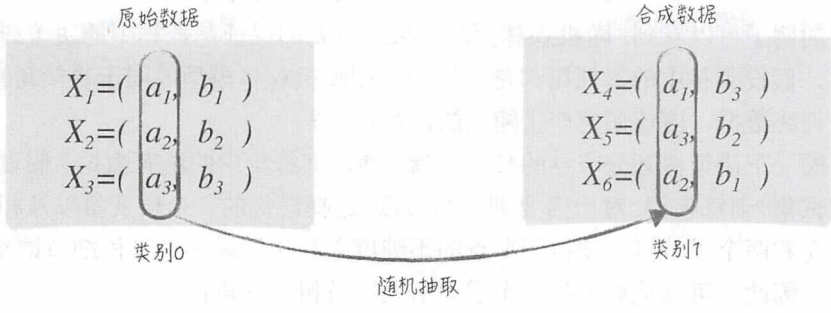


图 8-24

原始数据、合成数据以及生成的标签变量放在一起，就构成了有标签的训练数据，不妨记为$\{(\boldsymbol{X}_i, y_i), 1 \leqslant i \leqslant 2n\}$。对于这样的数据，可以对其使用随机森林进行分类，这样操作的目的有两个[38]。

- 分析各个变量之间是否存在比较强烈的相关关系。同原始数据相比，合成数据里每个变量的分布情况是没有变化的（因为随机抽取）。唯一不同的是，合成数据破坏了原始数据中各个变量间可能存在相关关系。因此当分类结果的误差较大时，说明原始数据的各个变量几乎是相互独立，反之则说明各变量间的相关关系比较强烈。
- 将原本较低维度的原始数据映射到高维空间，同支持向量学习机中的核函数类似。这在学术上被称为 random forest embedding，也是如此使用随机森林的主要目的。下面将讨论它的具体细节。

在 8.3.4 节中，我们讨论了如何利用决策树来做特征提取，而随机森林是由n棵决策树组成，因此相应的处理方法与之很相似。举个例子，假设训练好的随机森林（训练数据为

[37] 事实上，几乎所有的监督式学习都可以作为非监督学习的模型来使用，具体的方法与这里讨论的随机森林类似。

[38] 参考自“Manual for Setting Up, Using, and Understanding Random Forest V4.0”

书籍是人类进步的阶梯

$\{(\boldsymbol{X}_i, y_i), 1 \leqslant i \leqslant 2n\}$）里有两棵决策树，它们的叶子节点数分别为 2 和 4。第i个数据X_i落在了第一棵决策树的第 1 个叶子，第二棵决策树的第 2 个叶子。那么这个数据相应的新特征为(1, 0, 0, 1, 0, 0)，如图 8-25 所示，这样就完成了低维数据到高维数据（随机森林里决策树的个数可以很大）的映射。random forest embedding 方法常配合其他监督式学习模型一起使用，比如 9.2 节中将讨论的朴素贝叶斯（random forest embedding 的代码实现将在 9.2.6 节中讨论）。

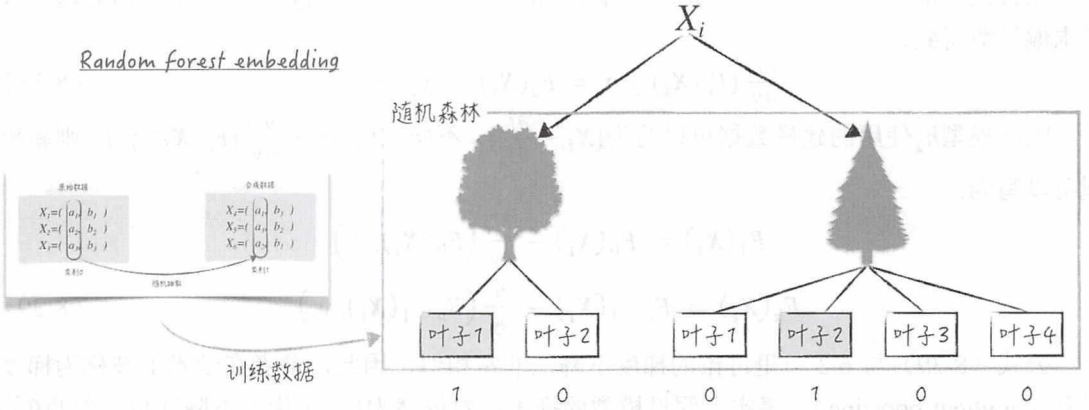


图 8-25

8.4.3 GBTs 之梯度提升

GBTs 表示 gradient-boosted trees[39]，虽然这是个很经典也很常用的模型，但没有正式的中文翻译名，我们不妨将它翻译成梯度提升决策树。为了更直观地理解模型，下面首先来看一个简单的回归问题。

假设有n个训练数据，记为$\{\boldsymbol{X}_i, y_i\}$。$\boldsymbol{X}, y$都是数值型变量，其中$X$表示自变量，而$y$表示标签变量（需要被预测的量）。假设现在已经有了最初的预测模型F_0（可以将其想象成简单的决策树），那么对于任意一点i，模型的残差（residual）为：

$$r_i = y_i - F_0(\boldsymbol{X}_i) \tag{8-35}$$

为了提升模型效果，现在对数据$\{\boldsymbol{X}_i, r_i\}$进行建模，得到模型$h_1$（也可以将它想象成决策树）。然后将这个模型和最初的模型组合成新的预测模型F_1：

$$F_1(\boldsymbol{X}_i) = F_0(\boldsymbol{X}_i) + h_1(\boldsymbol{X}_i) \tag{8-36}$$

数学上可以证明，相比与最初的模型F_0，新模型F_1的效果更好。事实上，如果模型h_1的

[39] 在某些文献中，这类模型也被称为 Gradient Boosting Decision Tree（GBDT）。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

预测结果是 100%准确，那么F_1也将 100%准确地预测y。如果新模型F_1的效果依然达不到要求，则不断重复上面的过程，得到新模型$F_k = F_{k-1} + h_k$。这个过程就是前面提到的 boosting（提升）。

现在将这个方法在数学上进行一次抽象，使得它能应用到更多的场景。针对上面的回归问题，它的损失函数为均方差（MSE），如公式（8-37）所示：

$$L = \frac{1}{2}\sum_i(\hat{y}_i - y_i)^2 \tag{8-37}$$

在公式（8-37）中，$L_i = 0.5(\hat{y}_i - y_i)^2$为$i$点对应的损失。将$\hat{y}_i$作为一个整体对这个式子求偏导数得到：

$$\frac{\partial L_i}{\partial \hat{y}_i}(F_0(\boldsymbol{X_i}), y_i) = F_0(\boldsymbol{X_i}) - y_i = -r_i \tag{8-38}$$

因此模型h_1使用的建模数据可以写为$\{\boldsymbol{X_i}, -\frac{\partial L_i}{\partial \hat{y}_i}\}$。不妨记$h_1 = -\frac{\partial \hat{L}_i}{\partial \hat{y}_i}(F_0(\boldsymbol{X_i}), y_i)$，则新模型可以写为：

$$F_1(\boldsymbol{X_i}) = F_0(\boldsymbol{X_i}) - \frac{\partial \hat{L}_i}{\partial \hat{y}_i}\left(F_0(\boldsymbol{X_i}), y_i\right)$$

$$F_k(\boldsymbol{X_i}) = F_{k-1}(\boldsymbol{X_i}) - \frac{\partial \hat{L}_i}{\partial \hat{y}_i}\left(F_{k-1}(\boldsymbol{X_i}), y_i\right) \tag{8-39}$$

公式（8-39）与 6.2 节里讨论的梯度下降法非常相似，因此，前者在学术上被称为梯度提升（gradient boosting）。事实上假设模型的损失函数依然为L，则梯度下降法里，参数θ的更新公式如下：

$$\theta_k = \theta_{k-1} - \eta\frac{\partial L}{\partial \theta} \tag{8-40}$$

仔细比较公式（8-39）和公式（8-40）可以得到如下内容。

- 梯度下降法使用的是损失函数对参数的真实梯度值，而梯度提升使用的是相应梯度的预测值。
- 在梯度下降法中，每次更新的是模型参数；而在梯度提升中，每次更新的是模型的预测值结果[40]。
- 数学上可以证明，梯度是损失函数下降速度最快的方向，因此梯度提升可以理解为以最优的方式提升模型的预测效果。

8.4.4 GBTs 之算法细节

在梯度提升的算法中使用决策树模型，就可以得到 GBTs。模型的具体算法如下（针对

[40] 仔细分析决策树会发现这个模型本身并没有明确的模型参数（因为决策树属于非参模型，没有固定数量的模型参数），因此无法用梯度下降法。而梯度提升很巧妙地跳过了这一点，把决策树本身抽象成整体模型的一个参数，并使用梯度下降的方法对其更新。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

回归问题），假设训练数据为$\{\boldsymbol{X}_i, y_i\}$。

（1）定义模型在数据点i的损失函数为$L(\hat{y}_i, y_i)$。

（2）定义初始模型为$F_0 = \sum_{i=1}^{n} y_i/n$，即被预测值的平均值。

（3）根据损失函数的梯度，得到新的训练数据$\{\boldsymbol{X}_i, -\frac{\partial L}{\partial \hat{y}_i}(F_0(\boldsymbol{X}_i), y_i)\}$，并用新的训练数据得到决策树模型$h_1$。

（4）更新模型得到$F_1 = F_0 + \gamma_1 h_1$，其中参数γ_1表示模型更新的力度。它也是 GBTs 算法的一部分，与梯度下降法中的学习速率（learning rate）是完全不同的[41]。它的估算公式为$\gamma_1 = \text{argmin}_\gamma \sum_{i=1}^{n} L(F_0 + \gamma h_1, y_i)$。

（5）不断重复上面的第（3）步和第（4）步，得到如公式（8-41）所示的模型更新公式。其中决策树h_k的训练数据为$\{\boldsymbol{X}_i, -\frac{\partial L}{\partial \hat{y}_i}(F_{k-1}(\boldsymbol{X}_i), y_i)\}$，而$\gamma_k = \text{argmin}_\gamma \sum_{i=1}^{n} L(F_{k-1} + \gamma h_k, y_i)$。

$$F_k = F_{k-1} + \gamma_k h_k \tag{8-41}$$

在实践中，常常事先选定 GBTs 的深度m（m为模型的超参数），即在模型中使用m个决策树，这种情况下，模型的预测公式为$F = F_0 + \sum_{i=1}^{m} \gamma_i h_i$。

用一个简单的例子来说明 GBTs 的学习过程，如图 8-26 所示。随着深度的不断增加，使用决策树不断地修正上一个模型遗留下来的错误，并以此提升 GBTs 的预测效果。

上面讨论的算法虽然是针对回归问题而设计的，但它可以很自然地应用到分类问题。比如针对二元分类问题，假设y_i等于 1 或者 0，如公式（8-42）所示定义分类的损失。这样分类问题就转变为了回归问题，可以用同样的算法解决。

$$L_i = L(\hat{y}_i, y_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i)\ln(1 + e^{\hat{y}_i}) \tag{8-42}$$

在具体的代码层面，第三方库 scikit-learn 提供了两种实现：GradientBoostingRegressor 解决回归问题，而 Gradient Boosting Classifier 解决分类问题。它们的调用过程和模型参数也与决策树类似，在此不再赘述。

---

[41] 在梯度下降法中，学习速率是一个超参数，在使用模型时人为进行设置，因此模型所用的学习速率并不是最优选择。而 GBTs 的算法中，我们计算得到了在当前梯度下的最优更新力度γ_n。它并不是模型的超参数，而是需要被估算的模型参数。

为了防止过拟合，在 GBTs 的工程实现上，同样会引入学习速率v这个超参数，这时模型的更新公式就变为：

$$F_k = F_{k-1} + v\gamma_k h_k$$

书籍是人类进步的阶梯

书籍是人类进步的阶梯

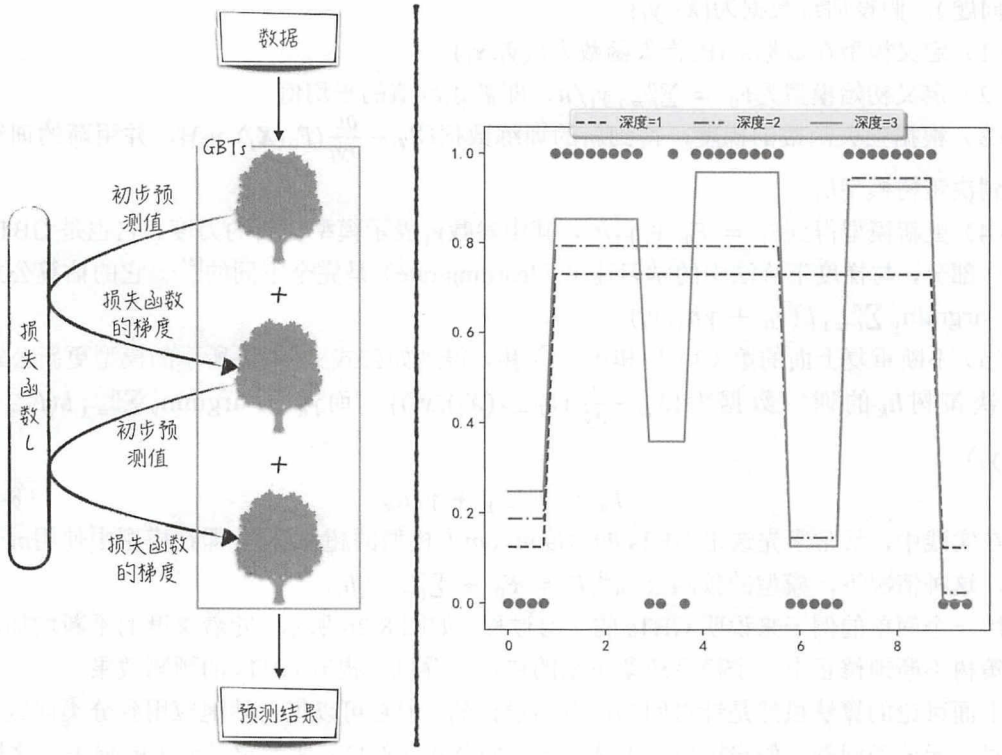


图 8-26[42]

8.5 本章小结

本章是讨论监督式学习的第一章，重点介绍了支持向量学习机、核函数、决策树、随机森林以及 GBTs。

支持向量学习机的出发点是解决线性可分或近似线性可分的问题。在这个模型中，有一个很重要的隐含假设：每个数据点的权重并不相同。除去少数几个支持向量（靠近分离超平面的数据），其他数据的权重其实等于 0。也就是说，支持向量学习机在训练时并不会考虑所有数据，而只关心其中很难被“直线”分开的“异常点”。

为了使支持向量学习机能处理非线性分类问题，学术界引入了核函数这个概念。核函数能够高效地完成空间变换，特别是从低维度空间到高维度空间的映射，能将原本非线性问题

[42] 完整的实现请参考随书配套的代码/ch08-supervised/decision_tree/GBTs.py。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

转换为高维空间里的线性问题。核函数是一个很通用的方法，在监督式和非监督式学习里都能见到它的身影。

接下来的决策树是一类很特殊的模型，它通过生成决策规则来解决分类和回归问题。整个模型非常容易理解，是所谓的白盒模型。但是由于决策树在理论上能无限制地划分节点，因此极易引起过拟合的问题。为了解决后者，决策树引入了剪枝的概念，其中包括前剪枝：在划分节点之前限制决策树的复杂度；以及后剪枝：在决策树构建完成之后，通过剪枝集修改树的结构，降低它的复杂度。这两个方法相比，前剪枝的实现更加容易，也更加可控，因此在实际中应用得更为广泛。

决策树最大的缺点在于模型的最后一步算法过于简单：对于分类问题，只考虑叶子节点里哪个类别占比最大；而对于回归问题，则计算叶子节点内数据的平均值。这导致它在单独使用时，预测效果不理想。因此在实际中，决策树常常被用来做特征提取，与其他模型联结起来使用。

为了提升树模型的效果，学术界引入了基于树的集成方法：将多个决策树按某种方式组合起来使用，从而提高模型的预测效果，其中最重要的两个代表分别是随机森林和 GBTs。随机森林的思路是模拟投票，按少数服从多数的原则产生最终的预测结果，它的理论基础是独立事件的概率法则。而 GBTs 则通过修正已有模型的错误来提升预测效果，它的理论基础是梯度下降法。

由于篇幅限制，本章只讨论了这些模型中最基础，也是最重要的部分，有很多模型细节很遗憾地没被涵盖到，包括解决回归问题的 SVR 模型（support vector regression）、决策树里的惩罚项、集成方法里的 AdaBoost 模型等。有兴趣的读者请参考其他书籍，比如 Trevor Hastie 等人编著的 *The Elements of Statistical Learning*。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

第9章

生成式模型：量化信息的价值

If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck.

（如果一个东西看起来像鸭子，游泳像鸭子，叫声像鸭子，那么它很可能就是鸭子。）

——美国俚语

9.1 贝叶斯框架

9.2 朴素贝叶斯

9.3 判别分析

9.4 隐马尔可夫模型

9.5 本章小结

书籍是人类进步的阶梯

书籍是人类进步的阶梯

本章讨论的模型仍属于监督式学习范畴，但与之前有所不同的是，这些模型都基于一种全新的建模理论：搭建模型的出发点不是为了对未知数据做预测，而是为了弄清楚并模拟数据产生的原理和机制。

为了更具体地讨论这种建模理论，我们假设数据的因变量为$\boldsymbol{X}$，被预测量为y。首先简单回顾一下之前讨论的监督式模型，比如第 5 章中介绍的逻辑回归（logistic regression），它直接从自变量$\boldsymbol{X}$出发，得到被预测量$y = 1$的概率，即$P(y = 1 \mid \boldsymbol{X}) = 1/(1 + \mathrm{e}^{-\boldsymbol{X}\boldsymbol{\beta}})$。之前讨论的其他模型跟逻辑回归的建模过程是一样的，也就是说，模型的建模出发点是直接考查自变量到被预测量的关系，即建模的对象是条件概率$P(y \mid \boldsymbol{X})$。在学术上，这些模型被称为判别式模型（discriminative model）。

而本章将讨论的是与之相对的生成式模型（generative model）。这类模型并不直接寻找自变量$\boldsymbol{X}$与被预测量y之间的因果关系，而是关心数据$\{\boldsymbol{X}, y\}$是如何产生的。也就是说，它的建模对象是条件概率$P(y)$和$P(\boldsymbol{X} \mid y)$。其中，$P(y)$表示被预测量的分布情况，它将解释数据y是如何产生的；而$P(\boldsymbol{X} \mid y)$表示在被预测量已知的情况下自变量的分布情况，它将解释数据$\boldsymbol{X}$是如何产生的。举个简单的例子，用$y = 1$表示鸭子，$y = 0$表示公鸡，则$P(\boldsymbol{X} \mid y = 1)$表示鸭子的叫声和样子等特征，而$P(\boldsymbol{X} \mid y = 0)$表示公鸡相对应的特征。

当然与判别式模型一样，生成式模型的最终目的是预测变量y。因此，生成式模型会在$P(\boldsymbol{X} \mid y)$和$P(y)$的基础上，根据贝叶斯框架得到可以用于做预测的条件概率$P(y \mid \boldsymbol{X})$。总结一下，生成式模型的建模流程为：首先假设在不同类别（事物的内在）下自变量（事物的表象）的分布情况，再根据观测到的实际情况，推导出隐藏在背后的类别，具体如图 9-1 所示。

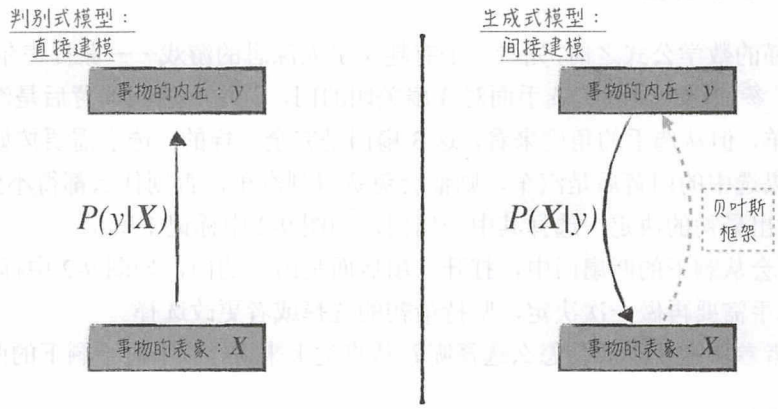


图 9-1

同判别式模型相比，生成式模型是搭建间接模型，因此到目前为止，它的模型效果往往不如判别式模型，而且模型的数学处理更为复杂。但生成式模型的建模理念非常诱人，不论

书籍是人类进步的阶梯

书籍是人类进步的阶梯

是学术界还是业界[1]都很看好它在未来的发展：它不但能根据特征预测结果，还能“理解”数据是如何产生的，并以此为基础“创造”数据，这才是“真正意义上”的人工智能。而且正如费曼[2]所说的“What I cannot create, I do not understand（我不能创造的东西，我就不了解）”，生成式模型在某种意义上是真正理解了数据。

本章将着重介绍3类模型：朴素贝叶斯、判别分析以及隐马尔可夫模型。朴素贝叶斯较为简单，各个特征是条件独立的。判别分析与朴素贝叶斯比较类似，只是它放宽了条件独立的假设，允许特征之间存在关联。隐马尔可夫模型则较为复杂，它以马尔可夫链为基础，通过引入隐含变量对序列数据进行建模。

9.1 贝叶斯框架

生成式模型的理论基础是贝叶斯定理[3]。这是一个简单而又深刻的数学定理：一方面，它只涉及乘法、除法以及条件概率，推导过程只需要用到高中数学；另一方面，它引出了很多极具哲学思辨色彩的学术概念，比如先验概率、后验概率等，甚至在统计学里也派生出相应的贝叶斯学派。

贝叶斯定理是深入理解生成式模型的关键，因此本节将深入讨论这个数学定理，以及如何基于它搭建模型。

9.1.1 蒙提霍尔问题

在讨论严谨的数学公式之前，先看一个有趣又引人深思的游戏——蒙提霍尔问题（Monty Hall problem）。参加这个游戏的选手面对3扇关闭的门，其中一扇门的背后是汽车，另外两扇的背后是山羊，但从选手的角度来看，这3扇门是完全一样的。选手需要按如下的步骤选中一扇门，如果选中的门背后是汽车，则他会免费得到汽车，否则什么都得不到。

- 选手做出最初的决定，选择其中一扇门，如图9-2中标记1所示。
- 主持人会从剩下的两扇门中，打开一扇后面是山羊的门，如图9-2中标记2所示。
- 这时选手需要再做一次决定，坚持最初的选择或者更改选择。

如果是读者参加这个游戏会怎么选择呢？从直觉上来讲，似乎对于剩下的两扇门，它们

[1] 由伊隆·马斯克（Elon Musk，美国企业家，他是PayPal、SpaceX以及特斯拉汽车的创始人）创建的人工智能公司OpenAI就极力推崇生成式模型。

[2] 理查德·菲利普斯·费曼（Richard Phillips Feynman），美国理论物理学家，量子电动力学创始人，曾被评选为有史以来最伟大的十位物理学家之一。

[3] 贝叶斯定理以英国数学家托马斯·贝叶斯（Thomas Bayes）的名字命名。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

背后是汽车的概率是相等的，都为 0.5，那么没必要修改自己的最初决定。

但遗憾的是，直觉是错误的：更改选择后的获奖概率远高于坚持最初决定的。事实上，借助 Python 针对蒙提霍尔问题做一个简单的统计模拟[4]，可以得到如图 9-2 中标记 4 所示的结果。根据模拟结果，更改后的获奖概率接近 70%，而坚持最初选择的获奖概率只有 30%左右。

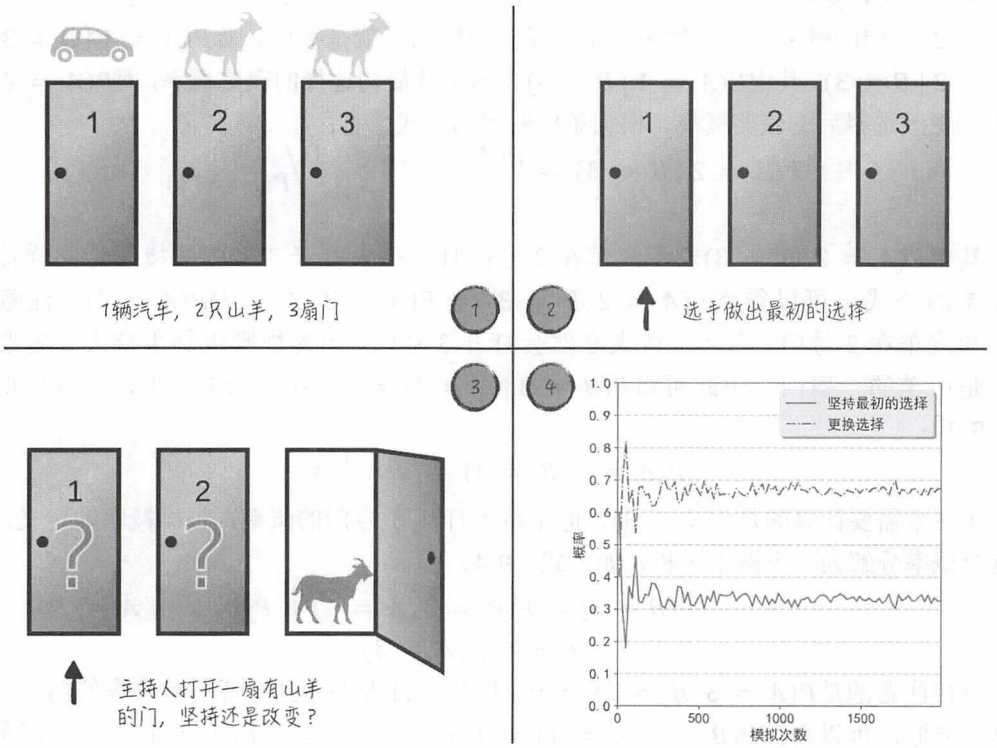


图 9-2[5]

为什么会出现如此奇怪的结果呢？直觉上哪里出了错呢？下面将从数学角度给出这个疑问的解释。

9.1.2 条件概率

为了表述清楚，不妨假设选手最初选择的是 1 号门，而剩下的 2 号门和 3 号门对主持人

[4] 这种统计模拟的方法在学术上被称为蒙特卡洛方法（Monte Carlo method）。该方法使用随机数（或者严谨地说是伪随机数）来解决很多复杂的概率计算问题。

[5] 图片参考自维基百科，图中标记 4 的具体实现请参考随书配套的代码/ch09-generative_models/monty_hall.py。

书籍是人类进步的阶梯

是没有任何差别的。用随机事件A表示汽车所在的位置，比如$A = 1$表示汽车在 1 号门的背后。那么在选手做最初选择的时候，汽车在每扇门背后的概率都是一样的，为$1/3$。

$$P(A = 1) = P(A = 2) = P(A = 3) = \frac{1}{3} \tag{9-1}$$

用随机事件B表示主持人打开的门号，比如$B = 3$表示主持人打开了 3 号门。由于 2 号门和 3 号门没有差别，那么只需讨论在已知主持人打开 3 号门的情况下，汽车在 1 号门以及汽车在 2 号门的概率。用条件概率来翻译上面这句话就是需要计算$P(A = 1 \mid B = 3)$以及$P(A = 2 \mid B = 3)$，其中$P(A = 1 \mid B = 3)$表示坚持最初选择的获奖概率，而$P(A = 2 \mid B = 3)$表示更改选择后的获奖概率。根据条件概率的公式，有：

$$P(A = 2 \mid B = 3) = P(A = 2, B = 3) \big/ P(B = 3) \tag{9-2}$$

其中$P(A = 2, B = 3)$表示汽车在 2 号门且主持人打开 3 号门的概率。同样根据条件概率的公式，可以得到$P(A = 2, B = 3) = P(B = 3 \mid A = 2)P(A = 2)$。注意到如果已知汽车在 2 号门，那么主持人必然会打开 3 号门，因为规则限制主持人一定要打开后面是山羊的一扇门。由此可得$P(B = 3 \mid A = 2) = 1$，结合公式（9-1），可以得到公式（9-3）：

$$P(A = 2, B = 3) = 1 \times \frac{1}{3} = \frac{1}{3} \tag{9-3}$$

接下来需要计算的是$P(B = 3)$，即主持人打开 3 号门的概率。根据概率的定义，可以将这个概率分解为 3 个概率之和，如公式（9-4）所示：

$$\begin{aligned} P(B = 3) = {} & P(B = 3, A = 1) + P(B = 3, A = 2) \\ & + P(B = 3, A = 3) \end{aligned} \tag{9-4}$$

值得注意的是$P(A = 3, B = 3) = 0$，因为主持人不会打开背后是汽车的门。与公式（9-3）类似，可以得到$P(B = 3, A = 1) = P(B = 3 \mid A = 1)P(A = 1)$。由于汽车在 1 号门时，主持人打开剩下两扇门的概率是一样的[6]，也就是说$P(B = 3 \mid A = 1) = 0.5$，那么$P(B = 3, A = 1) = 1/6$，结合公式（9-3）可以得到：

$$P(B = 3) = \frac{1}{3} + \frac{1}{6} = \frac{1}{2} \tag{9-5}$$

将公式（9-5）和公式（9-3）的结果代入公式（9-2），得到$P(A = 2 \mid B = 3) = 2/3$。也就是说更换选择后，获奖的概率为$2/3$，高于坚持最初选择的获奖概率。这与图 9-2 中标记 4 所示的模拟结果是吻合的。

---

[6] 这是因为前面假设了 2 号门和 3 号门对主持人没有任何差别。如果假设主持人对某扇门有特殊的偏好，比如在条件允许的范围内总是优先打开 3 号门，则条件概率不再适合，需要直接使用联合分布概率来解决这个问题（这也是蒙提霍尔问题最难理解的地方）。这种情况下，更换选择后的获奖概率为$P(A = 2, B = 3) + P(A = 3, B = 2) = 2/3$，具体的计算过程同公式（9-3）类似。

更直接地，对于主持人不同的开门策略，使用蒙特卡洛方法模拟，得到的结果是大体一致的。

将上面的数学推导用通俗的语言来解释，就是主持人打开 3 号门这个事件其实隐含了两条信息：

- 汽车在 1 号门后，主持人是随机打开 3 号门的；
- 汽车在 2 号门后，主持人并没有其他选择，只能打开 3 号门。

这两条信息对选手的最终选择有决定性的影响，而且是完全相反的影响。如果按照第 1 条信息，应该坚持最初的决定；如果按照第 2 条信息，则应该更换选择。

不仔细考虑的话，直觉上会认为这两条信息出现的频率是相等的，或者说这两条信息带来的价值是一样的。这会导致我们得到错误的结论：坚持最初决定和更改选择的获奖概率是一样的。为了克服直觉上的缺陷，需要使用合适的数学工具来量化信息带来的价值。而上面的数学推导表明，条件概率可以很好地完成这个任务，这正是贝叶斯框架的精妙之处。

9.1.3 先验概率与后验概率

现在从上面的具体例子回到抽象的数学。对于监督式学习，数据的变量分为自变量和标签变量（因变量）。自变量往往表示事物的表象，是很容易被观测到的，用$\boldsymbol{X}$表示。而标签表示事物的内在，不容易被观测到，也是模型需要预测的量，用y表示，则贝叶斯定理的数学公式为：

$$P(y \mid \boldsymbol{X}) = {P(\boldsymbol{X} \mid y)P(y)}\Big/{P(\boldsymbol{X})} \tag{9-6}$$

公式（9-6）中的$P(\boldsymbol{X} \mid y)$在学术上被称为先验概率（prior probability），而$P(y \mid \boldsymbol{X})$被称为后验概率（posterior probability）。通常可以这样理解，先验概率是用概率的形式表示生活中的常识。比如在蒙提霍尔问题中，在已知汽车在 2 号门背后的情况下，主持人打开各扇门的概率。而后验概率是通过事物的表象对产生原因的一种猜测，比如同样在蒙提霍尔问题中，在观察到主持人打开 3 号门的情况下，汽车在 2 号门背后的概率。用一句话来概括，先验概率是知因求果，后验概率是知果求因。

9.1.4 参数估计与预测公式

本节将讨论生成式模型的参数估计方法和预测公式。这部分内容比较抽象，但对理解生成式模型的运作原理很有帮助。

根据上面的讨论，我们知道生成式模型的建模对象是先验概率$P(\boldsymbol{X} \mid y)$和变量y的分布$P(y)$，由这两个概率可以得到变量$\boldsymbol{X}, y$的联合概率$P(\boldsymbol{X}, y) = P(\boldsymbol{X} \mid y)P(y)$，它表示当前数据出现的可能性，因此在生成式模型里，这个概率常被用作估计模型的参数。具体地，假设模型的参数为θ，定

义参数的似然函数（likelihood function）为$L = P(\boldsymbol{X}, y \mid \theta) = P(\boldsymbol{X} \mid y, \theta)P(y \mid \theta)$，则参数的估算公式为：

$$\hat{\theta} = \operatorname{argmax}_{\theta} P(\boldsymbol{X} \mid y, \theta)P(y \mid \theta) \tag{9-7}$$

这个方法在学术上被称为最大似然估计法（Maximum Likelihood Estimation，MLE），4.1.2节曾讨论过此方法。

得到模型参数的估计值之后，就可以根据公式（9-6）计算后验概率$P(y \mid \boldsymbol{X})$（为了书写简便，将$P(\boldsymbol{X} \mid y, \hat{\theta})$仍记为$P(\boldsymbol{X} \mid y)$，$P(y)$与$P(y \mid \boldsymbol{X})$也类似）。根据后验概率，可以很自然地得到$y$的预测公式，$\hat{y} = \operatorname{argmax}_y P(y \mid \boldsymbol{X})$，换句话说就是最大化后验概率。如果只是对变量$y$预测，其实并不需要计算公式（9-6）中的分母$P(\boldsymbol{X})$，由此可以得到：在实际生成中更为常用的简化版预测公式$\hat{y} = \operatorname{argmax}_y P(\boldsymbol{X} \mid y)P(y)$。

上面的预测公式不仅可以用于监督式学习，还可以用于非监督式学习。在后一类模型里，变量y不再是标签变量，而是不可观测的隐含变量（latent variable），表示造成事物表象的内在原因。这两类模型在数学处理上的差异如图9-3所示。

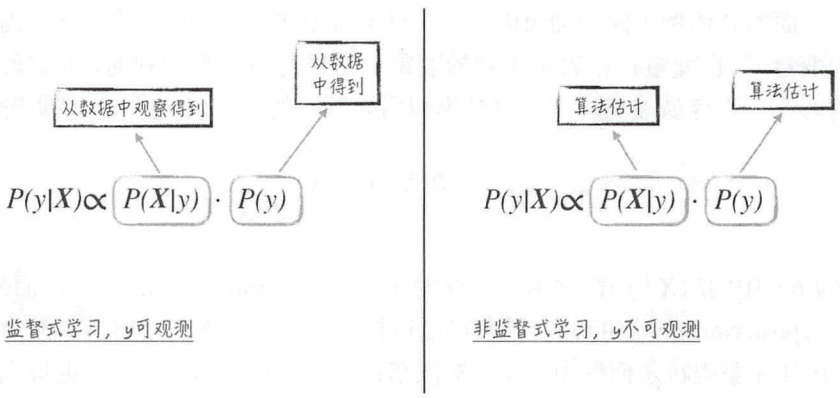


图9-3

在监督式学习中，由于变量y是可观测的，因此，相应的概率分布$P(y)$可以很容易地从数据中得到，类似地还有条件概率$P(\boldsymbol{X} \mid y)$。但在非监督式学习中，隐含变量y是不可观测的，因此只能在搭建模型时假设y的分布情况，并把它当作模型里的一个参数进行处理，也就是说需要用算法去“猜测”变量y的取值。类似地还有条件概率$P(\boldsymbol{X} \mid y)$，具体的处理方法将在介绍相应模型时再做详细讨论。

9.1.5 贝叶斯学派与频率学派

本节我们暂时脱离开具体的模型，探讨一个与生成式模型相关的哲学话题：贝叶斯学派

（Bayesian）与频率学派（frequentist）。这部分内容相对独立也比较发散，对方法论不感兴趣的读者可跳过此节。

贝叶斯学派和频率学派是统计学里彼此争论不休的两种理论体系，它们代表两种不同的建模出发点。简单来讲，这两个学派对“随机”这件事情的出发点和立足点是完全不同的。举一个具体的例子，在第 4 章中，我们讨论了线性回归模型。假设因变量为y，自变量为x，则线性回归模型的公式如下：

$$y = ax + b + \varepsilon \tag{9-8}$$

在公式（9-8）中，ε为随机扰动项，它的方差为σ^2，a, b, σ都为模型参数，是确定的值。数据中的随机性完全来源于随机扰动项。这其实是频率学派的建模方式：数据里的随机性是真实存在的，而且能被适合的模型所捕捉，另外这个模型的参数本身是确定的值。在求解模型时，参数的估计值是一个随机变量，它的随机性来源于数据的本身，可以通过假设检验或置信区间等统计工具来判断参数的估计值离真实值有多远。

在贝叶斯学派看来，线性回归模型[7]的数学表达虽然仍如公式（9-8）所示。但在这个公式里，参数a, b, σ不再是确定的值，而是随机变量。在具体的处理中，首先假设这些参数的先验分布，再根据数据得到参数的后验分布，如公式（9-9）所示，最后由参数的后验分布得到相应的估计值。

$$P(a, b, \sigma \mid y, x) \propto P(y \mid x, a, b, \sigma)P(a)P(b)P(\sigma) \tag{9-9}$$

从上面的公式可以看到，数据里的随机性更多地来自参数本身的随机分布。贝叶斯学派认为参数的分布情况其实反映了观察者对事件的认识并完美。观察者会根据不断得到的信息（训练模型的数据x, y），更新自己的知识（参数的后验分布），并最终根据新的知识，得到参数的估计值。

可以这样理解这两个学派的差异：频率学派更加“固执”，他们认为模型参数是不变的；而贝叶斯学派则更加“变通”，他们会根据新的信息，调整对模型参数的假设。从具体的方法上来看，这两个学派的差异可以用图 9-4 来表示。由于加入了参数的先验分布，贝叶斯学派的数学处理更为复杂，实用性并不是很强（虽然理论上更为严谨）。因此，这个学派的发展滞后于频率学派，以至于大部分统计书籍都以后者的理论为主。

[7] 这个模型在学术上被称为 Bayesian linear regression，是一种更加一般的线性回归模型。与普通的线性回归模型相比，此模型的参数估计值更加靠近 0。可以认为这个模型在线性回归的基础上加入了隐含的惩罚项，事实上，当参数的先验分布为特定分布时，此模型就是之前讨论过的岭回归（ridge regression，带有 L2 惩罚项的线性回归）。

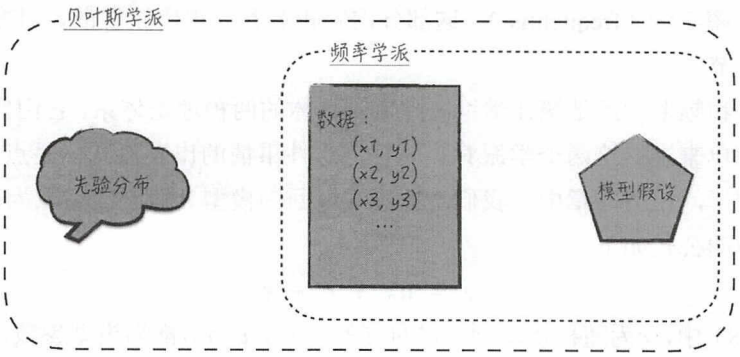


图 9-4

9.2 朴素贝叶斯

朴素贝叶斯（naive Bayes）是一类非常简单的解决分类问题的模型。与之前章节类似，假设数据有n个特征（自变量），记为$\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$；被预测量为$y$，表示类别。模型的假设非常简单，各个特征条件独立。换句话说，在同一类别下，各特征相互独立，如公式(9-10)所示。这个假设在学术上被称为 naive Bayes (NB) assumption。

$$P(x_1, x_2, \ldots, x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y) \tag{9-10}$$

朴素贝叶斯包含 3 个模型：伯努利模型、多项式模型以及高斯模型。其中高斯模型（Gaussian naive Bayes）可视为 9.3 节将讨论的二次判别分析的一个特例，因此这里先不讨论它的模型细节，而前两个模型常用于文本分类，比如将电子邮件分为垃圾邮件和正常邮件，又比如将报纸上的文章分为不同的主题等。下面就来看看这两个模型是如何工作的。

9.2.1 特征提取：文字到数字

文本分类属于自然语言处理（Natural Language Processing，NLP）范畴。它与之前章节讨论的模型有明显的不同：自然语言处理面对的是文字，而其他模型面对的是已经用变量表示的数据。

正如第 6 章和第 7 章中所讨论的，机器学习模型的本质是对数据的变量进行数学运算。这要求数据由一个或多个变量表示，而且变量能进行数学运算。也就是说，需要用数字变量去表示原始数据，更准确地说，用数字变量去描述原始数据所对应的物理实体。这恰好是文本分类或者自然语言处理中最难和最核心的部分。因为文本是由文字组成的，而对于人来讲，文字属于感性认识，数字属于理性认识，所以并没有特别“自然且有效”的方法将文字转换

为数字。当然一旦文字被有效地转换为数字变量，文本的分类问题就可以用之前讨论的分类模型解决了。

用一系列变量表示文本或文字在学术上被称为文字的特征提取。具体的方法有很多，也是人工智能研究的前沿和热门领域。这里先介绍其中最直接，也最简单的特征提取方法。

我们将可能出现的文字组成一个字典，并对字典进行排序[8]，比如“我”字排在第 1 位，用变量d_1表示；“成”字排在第 2 位，用变量d_2表示等。那么对于一个文本，用如下的向量$\boldsymbol{X}$（向量的长度等于字典的大小）来表示它：如果字典中排在i位的文字出现在文本中，则$x_i = 1$，否则$x_i = 0$。换句话说，变量x_i表示排在第i位的文字是否出现在当前文本里，整个过程如图 9-5 所示，比如“我爱你”这句话的特征表示为$x_1 = 1, x_{2001} = 1, x_{4092} = 1$，其他$x_i$等于 0。

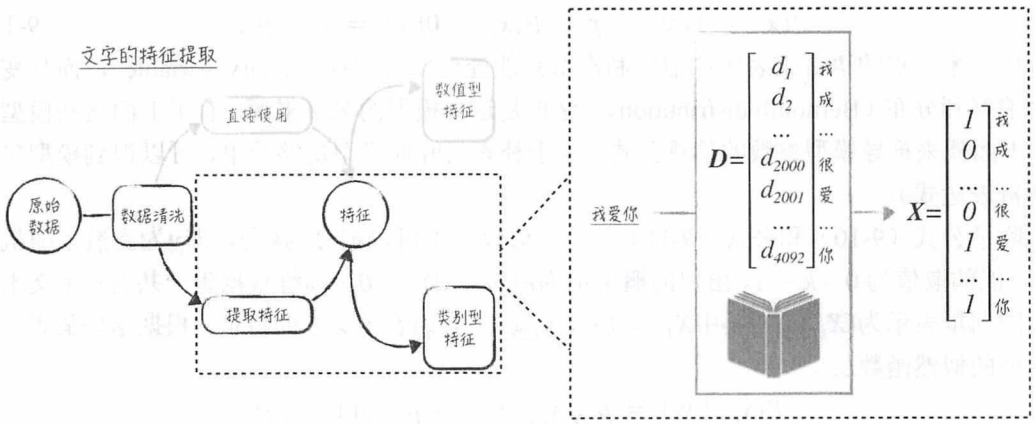


图 9-5

假设字典里一共有n个可能的文字，经过上面的处理，任何文本都能被n维向量所表示，这样就完成了文字的特征提取。值得注意的是，本章中提到的文字（或字）与实际生活中的文字含义不一样。实际生活中的文字表示单个汉字，而在本章中，为了表述简洁，用“文字”这个字眼表示学术上的语义单元（其含义接近于实际生活中所说的词），比如“我喜欢你”这句话中包含 3 个语义单元：“我”“喜欢”“你”。

[8] 在实际应用中，并不会把所有的文字都纳入字典中，通常会按如下的两步生成特征提取所依赖的字典。首先将训练文本中出现的文字组成初步的字典，然后在初步字典的基础上，剔除一些对文本分类没有帮助的文字，即所谓的停用词（stop words），比如“的”“地”“得”等。

针对中文的文本分类，在上面的步骤之前其实还有一步非常重要的处理，就是中文分词：将文本中的文字按意思分组，形成语义单元。同其他大部分字母文字不同，中文里的单个文字并不是一个语义单元，通常需要一个或多个文字组合在一起表示语义，比如“我喜欢你”这句话应该被分为“我”“喜欢”“你”3 个语义单元。这些分词后得到的语义单元将作为“文字”参与上面提到的字典生成。

仅供非商业用途或交流学习使用

9.2.2 伯努利模型

通过上面的特征提取算法，我们将文本转换为了n为向量$\boldsymbol{X}$，这个向量有两个特点。

- $\boldsymbol{X}$是一个非常稀疏的向量，因为相比与整个字典，一个文本中出现的文字是很有限的，所以它的大部分元素x_i都等于 0。
- $\boldsymbol{X}$的任何一个元素x_i都只有两个取值：0 或者 1。

这样的数据[9]正好符合伯努利模型（Bernoulli naive Bayes）的假设：除了朴素贝叶斯共有的 naive Bayes (NB) assumption（如公式（9-10）所示），它还假设数据的自变量都只取值 0 或者 1，而且出现的概率如公式（9-11）所示。

$$P(x_i = 1 | y) = p_{i,y};\ P(x_i = 0 | y) = 1 - p_{i,y} \tag{9-11}$$

用学术一点的语言来表述：伯努利模型只能处理二值变量（binary variable），而且变量符合伯努利分布（Bernoulli distribution）。这也是这个模型的名字来源。有了上面这些模型假设，下面就来推导模型参数的估算公式（由于朴素贝叶斯模型足够简单，可以得到模型参数的解析表达式）。

除了公式（9-10）和公式（9-11）外，还假设文本可以被分为k类，即y为离散型随机变量，可能的取值为 0～$k-1$，出现的概率记为$P(y = l) = \theta_l$；训练数据里一共有m个文本，相应的向量表示为$\{\boldsymbol{X}_i, y_i\}$，其中$\boldsymbol{X}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$表示文本的特征。根据这些假设，定义模型的似然函数$L$：

$$\begin{gathered} P(x_{i,j} \mid y_i) = p_{j,y_i} x_{i,j} + (1 - p_{j,y_i})(1 - x_{i,j}) \\ L_i = P(\boldsymbol{X}_i, y_i) = P(\boldsymbol{X}_i \mid y_i) P(y_i) = \prod_{j=1}^{n} P(x_{i,j} \mid y_i) P(y_i) \\ L = \prod_{i=1}^{m} L_i;\ \hat{\theta}_l, \hat{p}_{j,l} = \operatorname{argmax}_{\theta,p} L \end{gathered} \tag{9-12}$$

其中L_i表示文本i的概率。对于一个文本，在确定其类别的条件下，每个字出现与否是相互独立的，因此$L_i = \prod_{j=1}^{n} P(x_{i,j} \mid y_i) P(y_i)$。又由于文本之间是相互独立的，所以所有文本的联合概率为$L = \prod_{i=1}^{m} L_i$。经过一系列数学运算，可以得到如下的参数估计值：

$$\begin{gathered} \hat{\theta}_l = \sum_{i=1}^{m} 1_{\{y_i = l\}} \Big/ m \\ \hat{p}_{j,l} = \sum_{i=1}^{m} 1_{\{x_{i,j} = 1,\, y_i = l\}} \Big/ \sum_{i=1}^{m} 1_{\{y_i = l\}} \end{gathered} \tag{9-13}$$

[9] 针对高维数据（即自变量特别多的数据），大多数分类模型，比如逻辑回归，并不能很好地处理。因为数据的维度高意味着模型里的参数个数很多，而大多数分类模型的参数估计值并没有解析表达式，需要使用类似随机梯度下降法的最优化算法求解。对于后者，参数越多，算法的收敛速度也就越慢，这会导致模型在实际中几乎不可用。而伯努利模型正好是个例外，它可以在高维数据上，快速地求解出模型参数估计值，因此在实际中应用较广。

仅供非商业用途或交流学习使用

公式（9-13）其实非常简单，变量y的分布概率$\hat{\theta}_l$等于各类别在训练数据中的占比；而在已知文本类别的条件下，每个字的条件概率$\hat{p}_{j,l}$等于这个字在这个类别里的出现比例（出现的文本数除以这个类别的总文本数），如图 9-6 所示。有了模型参数的估算公式，就可以得到对文本i的最终预测结果：

$$\hat{y}_i = \operatorname{argmax}_l \prod_{j=1}^{n} P(x_{i,j} \mid y_i = l) P(y_i = l) \tag{9-14}$$

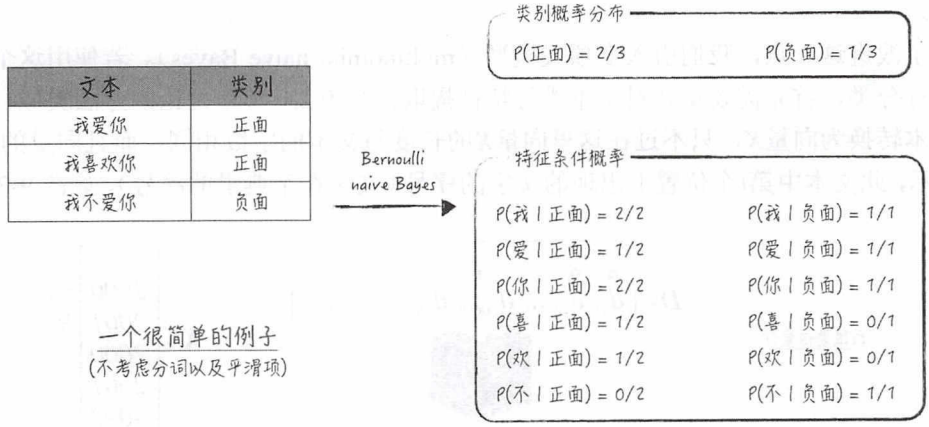


图 9-6

上面的推导虽然在理论上是比较完美的，但在实际应用中常常会遇到问题。考虑下面这种常见情况，假设字典里第t位是一个生僻字“兣”，它在训练文本里从来没有出现过。根据公式（9-13），得到对于所有的$l \in [0, k-1]$，都有$\hat{p}_{t,l} = 0$。如果需要预测的文本i中出现了这个字，那么$P(x_{i,t} \mid y) \equiv 0$，这样公式（9-14）中右边的乘积都等于 0，没办法得到正确的预测结果。

为了规避这个问题[10]，我们引入所谓的平滑项（Laplace/Lidstone smoothing）。将每个字的条件概率[11]估算公式改为公式（9-15），即在原有公式的基础上，分母加上2α，分子加上α。其中，$0 < \alpha \leqslant 1$，被称为平滑系数；$\alpha = 1$对应着 Laplace smoothing；$\alpha < 1$对应着 Lidstone smoothing。

$$\hat{p}_{j,l} = \left(\sum_{i=1}^{m} 1_{\{x_{i,j}=1, y_i=l\}} + \alpha\right) \Big/ \left(\sum_{i=1}^{m} 1_{\{y_i=l\}} + 2\alpha\right) \tag{9-15}$$

[10] 这个问题在某种程度上是过拟合问题，在训练数据中没有出现某个字就认为它出现的概率等于 0，这会使模型对训练数据的效果很好，但显然是不合理的结果。

[11] 在实际应用中，通常不会对每个类别的出现概率使用 Laplace smoothing（即使使用了，对结果的影响也不大），因为每个类别都会在训练数据中出现，而且出现的次数都比较多。

仅供非商业用途或交流学习使用

9.2.3 多项式模型

仔细分析伯努利模型的思路会发现，这个模型的基本假设是一个文本的主题只与某些字的出现与否有关。这当然是有一定道理的，但仍有许多不合理的地方。通常来说，在同一个文本中，一个字出现的次数越多，它与文本的主题也就越相关，这一点是伯努利模型没办法捕捉的。

为了改进这一点，我们引入多项式模型（multinomial naive Bayes）。若使用这个模型对文本进行分类，首先需要重新对文本进行特征提取：与 9.2.1 节中讨论的方法类似，借助字典将文本转换为向量$\boldsymbol{X}$，只不过在这里向量$\boldsymbol{X}$的长度与文本的字数相同，而且向量的第i个元素x_i表示，此文本中第i个位置上出现的文字的序号（文字在字典里的序号），如图 9-7 所示。

图 9-7

对于变量x_i的条件概率，多项式模型有如下的假设：

$$P(x_i = k \mid y) = p_{k,y} \tag{9-16}$$

也就是说，在文本类别已知的情况下，文本第i个位置出现某个字的概率只与字有关，与位置无关，比如图 9-7 中，第 2 个位置和第 4 个位置都出现了“很”字（$k = 2000$），则$P(x_2 \mid y) = P(x_4 \mid y) = p_{2000,y}$。

对于变量y的分布，多项式模型同样假设$P(y = l) = \theta_l$。与公式（9-12）的计算过程类似，可以得到相应模型参数的估算公式，其中，m为训练数据中文本的个数，$\boldsymbol{X_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$表示第$i$个文本的向量表示。

$$\hat{\theta}_l = \sum_{i=1}^{m} 1_{\{y_i = l\}} \Big/ m$$

$$\hat{p}_{k,l} = \sum_{i,j} 1_{\{x_{i,j} = k, y_i = l\}} \Big/ \sum_k \sum_{i,j} 1_{\{x_{i,j} = k, y_i = l\}} \tag{9-17}$$

公式（9-17）看着比较复杂，其实表示的意思非常简单，变量y的分布概率$\hat{\theta}_l$等于各类别在训练数据中的占比；而在已知文本类别的条件下，字典中第k个字的条件概率$\hat{p}_{k,l}$等于这个字出现的次数占这个类别总字数的比例，如图 9-8 所示。

仅供非商业用途或交流学习使用

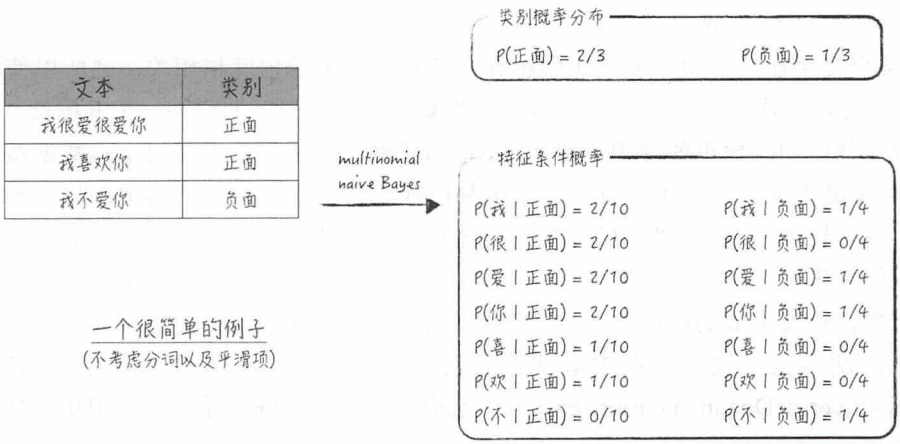


图 9-8

有了如公式（9-17）所示的参数估计，模型对变量y的预测公式与公式（9-14）类似，在此就不再赘述。同样地，与伯努利模型类似，为了规避生僻字带来的干扰，在公式（9-17）的基础上引入平滑项（Laplace/Lidstone smoothing），得到公式（9-18）。其中，n表示字典的大小，参数α为平滑系数，其含义与伯努利模型里的一致。

$$\hat{p}_{k,l} = \left(\sum_{i,j} 1_{\{x_{i,j}=k,\, y_i=l\}} + \alpha\right) \Big/ \left(\sum_k \sum_{i,j} 1_{\{x_{i,j}=k,\, y_i=l\}} + \alpha n\right) \tag{9-18}$$

9.2.4 TF-IDF

需要注意的是，9.2.3 节中所讨论的文本特征提取方法（如图 9-7 所示）以及模型推导都只适用于理论演算。在实际应用中，我们会把文本转换为n维向量X（假设字典大小为n），向量中的第i个元素x_i表示第i个文字在文本中出现的次数。而模型参数的估算公式改为（不考虑平滑项）：

$$\hat{\theta}_l = \sum_{i=1}^{m} 1_{\{y_i=l\}} \Big/ m$$

$$\hat{p}_{k,l} = \sum_{i=1}^{m} x_{i,k} 1_{\{y_i=l\}} \Big/ \sum_k \sum_{i=1}^{m} x_{i,k} 1_{\{y_i=l\}} \tag{9-19}$$

假设第i个文本的向量表示为$\boldsymbol{X}_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})$，则该文本的条件概率为：

$$P(\boldsymbol{X}_i \mid y = l) = \hat{\theta}_l \prod_{k=1}^{n} \hat{p}_{k,l}{}^{x_{i,k}} \tag{9-20}$$

仔细分析公式（9-19）和公式（9-20）会发现，多项式模型假设，对于第i个文本，它的类别几乎只与文字出现的绝对次数$x_{i,k}$相关，这个假设显然不太合理。实际上文字对文本主

仅供非商业用途或交流学习使用

题的影响可以分为如下两部分。

- 某个文字在文本中出现的比例越高，它与文本主题也就越相关。之所以使用文字出现比例，而非出现次数是因为文字的出现次数受文本自身长度的影响。文本越长，那么某个文字出现在其中的次数也就会相应地增多。使用比例在某种程度上是对文本数据做了归一化。
- 如果某个文字在几乎所有的文本中都出现，那么说明它是常用字，与文本主题的相关性并不高，比如“首先”“然后”等。反过来，如果某个字只在少数几个文本中出现，那么它很可能是表示文本主题的专业名词，比如“射门”“投篮”等。总结一下，一个文字出现的文本数占总文本数的比例越小，它与文本主题也就越相关。

上面的两点就是文本处理中最常用的文字加权技术，在学术上称为 TF-IDF（Term Frequency–Inverse Document Frequency）。上面的第一点是 TF，第二点是 IDF，具体的数学公式[12]如下：

$$
\begin{aligned}
TF_{i,k} &= {x_{i,k}}/{\sum_k x_{i,k}} \\
IDF_k &= \ln(m/\textstyle\sum_i 1_{\{x_{i,k}>0\}}) \\
TFIDF_{i,k} &= TF_{i,k} IDF_k
\end{aligned}
\tag{9-21}
$$

在实际应用中，常常对文本向量进行 TF-IDF 变换后再使用多项式模型[13]，这样能提升模型的分类效果。

9.2.5 文本分类的代码实现[14]

上面主要讨论了模型的理论细节，现在来看看如何使用 Python 实现模型，并用搭建好的模型对文本进行分类。

本节使用的文本数据来源于复旦大学计算机信息与技术系国际数据库中心自然语言处理小组（由复旦大学李荣陆提供），数据存储结构如图 9-9 所示。这份数据中一共有 20 个类别，为了突出重点，我们只选取其中较多的 4 个类别进行建模。它们分别是“C3-Art”（艺术）、“C11-Space”（太空）、“C19-Computer”（电脑）以及“C39-Sports”（体育）。

借助第三方库 scikit-learn，具体的代码实现并不困难。首先讨论伯努利模型的实现，如程序清单 9-1 所示。

---

[12] 这里只给出了 TF-IDF 最简单，也是最经典的定义。在实际应用中，往往会在公式（9-21）的基础上加入惩罚项和平滑项。但这些内容过于细节，且与本章主题无关，在此不再赘述。

[13] 虽然多项式模型在搭建模型时假设，向量X的元素x_i为正整数，表示文字出现的次数。但从数学上来看，无论是模型参数的估计公式（公式（9-19）），还是模型的预测公式（公式（9-20）），都只需x_i为正实数即可。这也是多项式模型能结合 TF-IDF 使用的原理所在。

[14] 完整的实现请参考随书配套的代码/ch09-generative_models/naive_bayes/text_classification.py。

仅供非商业用途或交流学习使用

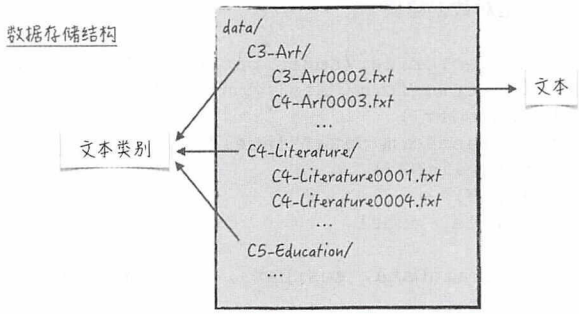


图 9-9

（1）正如 9.2.1 节中讨论的，文本分类的第一步是对文本进行特征提取。这一步可以由 scikit-learn 提供的 CountVectorizer 来完成，如第 9、10 行代码所示。这个类将把输入的字符串拆分成一个一个的字（默认情况下，分隔符为空格），并对每个字在文本内进行计数，具体的运行效果如图 9-5 所示。

（2）CountVectorizer 的参数“token_pattern”是一个正则表达式，它将决定什么样的字符串将作为字被纳入字典中，第 9 行代码中的“r"(?u)\b\w+\b"”表示任何长度等于 1 的字符串都将作为字。参数“binary=True”表示生成的实例“vect”将只统计文字出现与否，而“binary=False”表示“vect”将统计文字出现的次数。

（3）由于文本的类别也是文字，因此也需要将其转换为数字，如第 11、12 行代码所示，LabelEncoder 可以实现这个功能。

（4）BernoulliNB 类实现了伯努利模型，使用这个类对转换后数据进行建模，如第 13、14 行代码所示。这个类里有参数“alpha”，此参数就是平滑系数，具体的含义请参考公式(9-15)。

（5）在使用模型对未知数据进行预测时，首先需要对文本进行特征提取，再使用训练好的模型继续预测，如第 23 行代码所示，其中“vect.transform(testDocs)”表示对测试文本进行特征提取，使其变成向量（“testDocs”为字符串数组）。

（6）从代码的角度来讲，程序清单 9-1 这样的实现并不理想。因为这样并没有将模型的全部细节封装起来。使用模型时，仍然需要按步骤复现整个变换过程，没办法做到只调用一个 API 就得到预测结果（事实上，scikit-learn 已提供了相应的方法 Pipeline 来实现这样的效果，具体的细节将在后面讨论）。

程序清单 9-1 朴素贝叶斯之伯努利模型

```
1 | from sklearn.feature_extraction.text import CountVectorizer
2 | from sklearn.naive_bayes import BernoulliNB
3 | from sklearn.preprocessing import LabelEncoder
4 |
5 | def trainBernoulliNB(data):
6 |     """
```

仅供非商业用途或交流学习使用

```
    使用伯努利模型对数据建模
    """
    vect = CountVectorizer(token_pattern=r"(?u)\b\w+\b", binary=True)
    X = vect.fit_transform(data["content"])
    le = LabelEncoder()
    Y = le.fit_transform(data["label"])
    model = BernoulliNB()
    model.fit(X, Y)
    return vect, le, model

def trainModel(trainData, testData, testDocs, docs):
    """
    对分词后的文本数据分别使用多项式和伯努利模型进行分类
    """
    # 伯努利模型
    vect, le, model = trainBernoulliNB(trainData)
    pred = le.classes_[model.predict(vect.transform(testDocs))]
    ### # 略去后面的代码
```

在不分词的情况下（也就是说，单个汉字作为语义单元被纳入字典），运行上面的代码，可以得到如图 9-10 中标记 1 所示的结果。伯努利模型的分类效果很一般，查准查全率都在80%左右；对示例文本的分类结果也是错误的，比如“前国际米兰巨星雷科巴正式告别足坛”是一个典型的体育文本，但模型错误地将其归类为“C11-Space”（太空）。

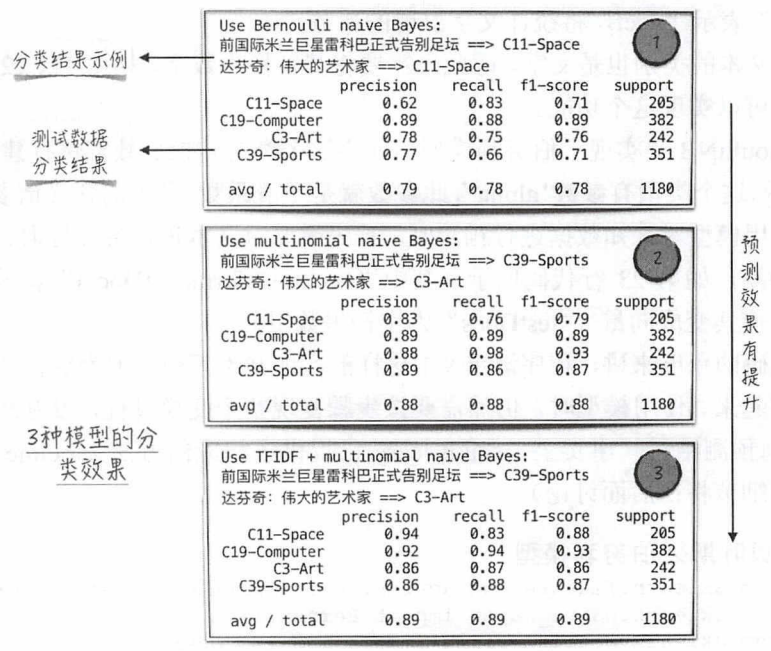


图 9-10

仅供非商业用途或交流学习使用

接下来将讨论多项式模型的实现。这里我们将接触到一个很重要的概念——Pipeline（流水线）。它的目的是在机器学习中如何将各种不同的模型组装起来使用，可以认为是模型联结主义的工程实现（有关模型联结的细节请参考 7.6.5 节）。具体的细节如程序清单 9-2 所示。

（1）多项式模型实现步骤与伯努利模型类似，首先使用 CountVectorizer 实现文本到向量的转换，再使用 MultinomialNB 对转换后的数据进行建模。因此可以使用 scikit-learn 提供的 Pipeline 将这两个不同的模型类“粘起来”形成一个整体，如第 10、11 行代码所示。

（2）Pipeline 用 Python 中的 tuples 类来表示模型里的一个步骤，比如“("model", MultinomialNB())”表示步骤的名字为“model”，相应的实例为“MultinomialNB()”。将这些步骤按顺序组成一个 list 传给 Pipeline 就完成了模型的联结。值得特别注意的是，对于多项式需要统计文字在文本中出现的次数，因此在 CountVectorizer 里设置“binary=False”（默认情况）。

（3）使用 Pipeline 之后，训练模型和使用模型就非常方便了，训练模型只需直接调用 fit 方法，如第 14 行代码所示；而使用模型只需直接调用 predict 方法，如第 24 行代码所示。

程序清单 9-2 朴素贝叶斯之多项式模型

```
1  | from sklearn.feature_extraction.text import CountVectorizer
2  | from sklearn.naive_bayes import MultinomialNB
3  | from sklearn.preprocessing import LabelEncoder
4  | from sklearn.pipeline import Pipeline
5  |
6  | def trainMultinomialNB(data):
7  |     """
8  |     使用多项式模型对数据进行建模
9  |     """
10 |     pipe = Pipeline([("vect", CountVectorizer(token_pattern=r"(?u)\b\w+\b")),
11 |         ("model", MultinomialNB())])
12 |     le = LabelEncoder()
13 |     Y = le.fit_transform(data["label"])
14 |     pipe.fit(data["content"], Y)
15 |     return le, pipe
16 |
17 | def trainModel(trainData, testData, testDocs, docs):
18 |     """
19 |     对分词后的文本数据分别使用多项式和伯努利模型进行分类
20 |     """
21 |     ### # 略去前面的代码
22 |     # 多项式模型
23 |     le, pipe = trainMultinomialNB(trainData)
24 |     pred = le.classes_[pipe.predict(testDocs)]
25 |     ### # 略去后面的代码
```

在不分词的情况下，使用多项式模型对文本分类可以得到如图 9-10 中标记 2 所示的预测结果。可以看到，模型效果相比于伯努利模型有了较大幅度的提升，对示例文本的分类也

仅供非商业用途或交流学习使用

是正确的。

正如 9.2.4 节中所讨论的，在多项式模型之前，可以使用 TF-IDF 对文本向量做进一步的“归一化”运算。具体的实现代码很简单，只需在 Pipeline 中加入相应的类 TfidfTransformer，如程序清单 9-3 中第 10～12 行代码所示。模型的分类结果如图 9-10 中标记 3 所示，分类效果有了进一步的提升。

程序清单 9-3 朴素贝叶斯之 TF-IDF + 多项式模型

```
1 | from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
2 | from sklearn.naive_bayes import MultinomialNB
3 | from sklearn.preprocessing import LabelEncoder
4 | from sklearn.pipeline import Pipeline
5 |
6 | def trainMultinomialNBWithTFIDF(data):
7 |     """
8 |     使用 TFIDF+多项式模型对数据建模
9 |     """
10 |     pipe = Pipeline([("vect",
CountVectorizer(token_pattern=r"(?u)\b\w+\b")),
11 |         ("tfidf", TfidfTransformer(norm=None, sublinear_tf=True)),
12 |         ("model", MultinomialNB())])
13 |     le = LabelEncoder()
14 |     Y = le.fit_transform(data["label"])
15 |     pipe.fit(data["content"], Y)
16 |     return le, pipe
```

在机器学习领域，对中文的处理还有一个很重要的环节，就是分词：把一个或几个字组合在一起形成语义单元。第三方库 jieba（结巴中文分词）是目前为止最成熟的中文分词工具。我们借助它对文本进行分词，再在此基础上使用同样的模型对文本进行分类，可以得到如图 9-11 所示的结果。结果显示，分词之后，各个模型的预测准确度均有了大幅提升。

分词的代码实现非常简单，直接调用方法 jieba.cut[15]即可，比如“jieba.cut("我喜欢你")”。其实，中文分词本身也是生成式模型很常见的应用场景，比如 jieba 就使用了隐马尔可夫模型（HMM），关于此模型的细节将在之后的 9.4 节中详细讨论。

到此为止，我们主要讨论了朴素贝叶斯模型（伯努利模型和多项式模型）在文本分类领域的应用。从分类的结果上来看，虽然模型很简单，但实际的分类效果其实并不差，查准查全率都在 96%左右。当然，这并不是效果最好的文本分类算法，比如同样是生成式模型的 LDA（Latent Dirichlet Allocation），它的分类效果就更好。但与这些模型相比，朴素贝叶斯足够简单，而且可解释性更强。

[15] 在使用 jieba 之前，需在计算机上安装好它。具体的安装命令为“pip install jieba”或“easy_install jieba”。

仅供非商业用途或交流学习使用

分类结果示例

测试数据分类结果

```
Use Bernoulli naive Bayes:
前国际米兰巨星雷科巴正式告别足坛 ==> C11-Space
达芬奇：伟大的艺术家 ==> C19-Computer
                precision    recall  f1-score   support
    C11-Space        0.89      0.84      0.86       205
C19-Computer         0.91      0.99      0.94       382
      C3-Art         0.94      0.77      0.85       242
   C39-Sports        0.84      0.89      0.86       351

 avg / total         0.89      0.89      0.89      1180
```

1

```
Use multinomial naive Bayes:
前国际米兰巨星雷科巴正式告别足坛 ==> C39-Sports
达芬奇：伟大的艺术家 ==> C3-Art
                precision    recall  f1-score   support
    C11-Space        0.97      0.89      0.93       205
C19-Computer         0.96      0.96      0.96       382
      C3-Art         0.96      0.99      0.97       242
   C39-Sports        0.96      0.97      0.97       351

 avg / total         0.96      0.96      0.96      1180
```

2

对中文分词后，3种模型的预测准确度均有大幅提升

```
Use TFIDF + multinomial naive Bayes:
前国际米兰巨星雷科巴正式告别足坛 ==> C39-Sports
达芬奇：伟大的艺术家 ==> C3-Art
                precision    recall  f1-score   support
    C11-Space        0.98      0.92      0.95       205
C19-Computer         0.97      0.96      0.96       382
      C3-Art         0.94      0.98      0.96       242
   C39-Sports        0.95      0.97      0.96       351

 avg / total         0.96      0.96      0.96      1180
```

3

预测效果有提升

图 9-11

9.2.6 模型的联结

除了文本分类，朴素贝叶斯还可以和其他模型联结，用于解决其他分类问题。现在以伯努利模型为例，讨论这种方法的细节。

与其他分类模型相比，伯努利模型的优点在于它很擅长处理高维的稀疏数据，但它的缺点也同样明显，它只能处理二值变量。因此，若数据里存在数值型变量（定量变量），需要通过划分区间的方式，将其转换为虚拟变量（dummy variable）使用，具体的细节请参考 7.3 节。当然对于类别型变量（定性变量），也需要将其转换为虚拟变量，这和其他模型一致。但如果仅这样处理，模型的预测效果并不好，因为通常情况下，伯努利模型并不擅长处理低维数据。为了提升模型效果，需要借鉴支持向量学习机（SVM）中核函数（kernel method）的做法，将低维数据映射到高维空间，而且需要保证在高维空间里，各变量的取值只能是 0 或者 1。8.4.2 节所讨论的 random forests embedding 正好满足这样的要求，因此在实际中，常使用它将数据升到高维，再在此基础上使用伯努利模型对数据进行分类。

举个简单的例子，假设原始数据只有两个变量x_1, x_2，将数据表现在平面上，可以得到

如图 9-12 所示的图形，图中的三角形表示类别 0，而圆点表示类别 1，两个类别均呈半月型。如程序清单 9-4 所示，先使用 RandomTreesEmbedding 将原始的二维数据映射到高维空间，再使用 BernoulliNB 对变换后的数据做分类。分类的结果还不错，图 9-12 中灰色部分的预测结果是类别 0 的区域，而白色区域的预测结果是类别 1。

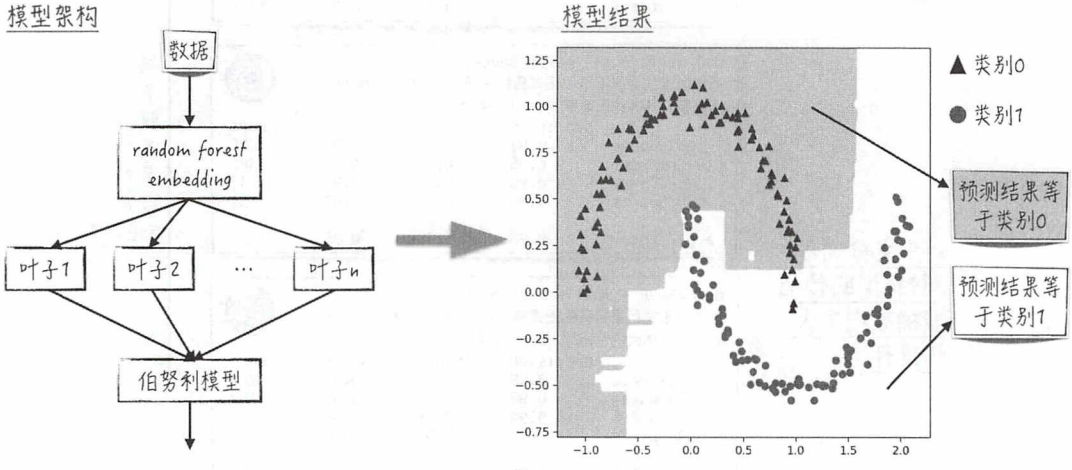


图 9-12[16]

程序清单 9-4 朴素贝叶斯之 random forest embedding + 伯努利

```
from sklearn.ensemble import RandomTreesEmbedding
from sklearn.naive_bayes import BernoulliNB
from sklearn.pipeline import Pipeline

def trainModel(data):
    """
    使用 random forest embedding+伯努利模型对数据建模
    """
    pipe = Pipeline([("embedding", RandomTreesEmbedding(random_state=1024)),
        ("model", BernoulliNB())])
    pipe.fit(data[["x1", "x2"]], data["y"])
    return pipe
```

9.3 判别分析

本节将讨论如何在贝叶斯框架下，借助正态分布对数据建模。在学术上，这类模型被称

[16] 完整的实现请参考随书配套的代码/ch09-generative_models/naive_bayes/classification_example.py。

为判别分析[17]（discriminant analysis）。与朴素贝叶斯相比，判别分析最大的改进在于它放弃了条件独立这个假设，允许变量间存在关联关系，这使得它的适用场景更多。

判别分析主要包括两个模型：线性判别分析和二次判别分析。下面将讨论它们的模型细节。

9.3.1 线性判别分析

线性判别分析（Linear Discriminant Analysis，LDA）是解决分类问题的模型。与之前讨论的伯努利模型和多项式模型相反，它只能处理连续型变量。

为了数学上表述简洁，我们不妨设训练数据只有两个自变量，记为$\boldsymbol{X} = (x_1, x_2)$，而且数据类别也只有两个，记为$y = 0$或者$y = 1$。线性判别的数学假设如下：

$$\boldsymbol{X} \mid y = 0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma})$$
$$\boldsymbol{X} \mid y = 1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma})$$
$$P(y = 0) = \theta_0, P(y = 1) = \theta_1 \tag{9-22}$$

公式（9-22）中的$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$表示二元正态分布，其中$\boldsymbol{\mu}$是二维行向量表示分布的中心，而$\boldsymbol{\Sigma}$为$2 \times 2$的对称矩阵[18]表示分布的协方差（covariance）。根据上面的公式，有以下几点值得注意。

- 模型假设，在类别已知的条件下，自变量X服从正态分布。这就要求自变量表示的是一个能连续变化的量。有一些数值型变量（定量变量）并不满足这样的要求，比如在一个文本里，每个文字出现的次数，它就不是一个连续变化的量。
- 模型还假设，对于不同的类别，自变量的协方差是一样的，只是期望不一样。对于正态分布，协方差决定了分布的形状，而期望只决定分布的中心位置，如图 9-13 所示。因此从直观上可以这样理解，线性判别分析只区分各类别中心位置的差异，而不关心不同类别具体形状上的差别（事实上，模型假设不同类别的分布形状是一样的）。
- 数学上可以证明，若协方差$\boldsymbol{\Sigma}$是一个对角矩阵（也就是说除了对角线，其他元素都等于 0），则自变量x_1和x_2是相互独立的。反之，则这两个变量是相互关联的。因此，线性判别分析并没有变量条件独立这个假设。

与朴素贝叶斯类似，数学上可以推导出线性判别分析模型参数的表达式，如公式（9-23）所示，其中，m表示训练数据集的大小。具体的数学演算过程在这里就不展开了。

---

[17] 需要注意的是，虽然模型的名字是判别分析，但它是生成式模型，不是判别式模型。之所以叫这个有点奇怪的名字，是因为该类模型是由统计上的费舍尔线性判别方法（Fisher's linear discriminant）发展而来。

[18] 数学上严谨的表述是：$\boldsymbol{\Sigma}$为二阶半正定矩阵（positive semi-definite matrix）。

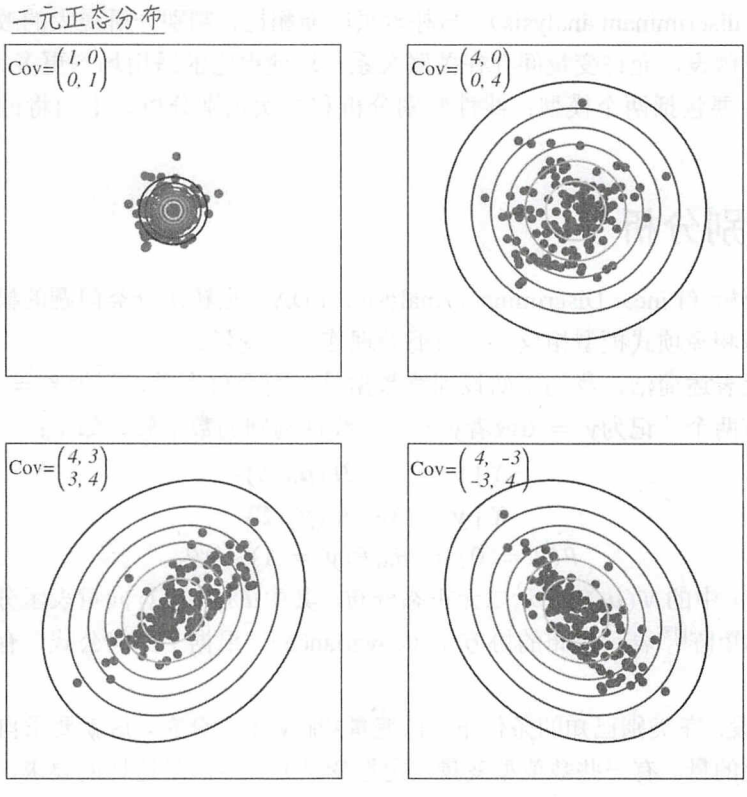


图 9-13[19]

$$\hat{\theta}_l = \frac{1}{m}\sum_{i=1}^{m} 1_{\{y_i = l\}}$$

$$\hat{\boldsymbol{\mu}}_l = \sum_{i=1}^{m} \boldsymbol{X}_i 1_{\{y_i = l\}} \Big/ \sum_{i=1}^{m} 1_{\{y_i = l\}}$$

$$\hat{\boldsymbol{\Sigma}} = \frac{1}{m}\sum_{i=1}^{m} (\boldsymbol{X}_i - \hat{\boldsymbol{\mu}}_{y_i})^{\mathrm{T}} (\boldsymbol{X}_i - \hat{\boldsymbol{\mu}}_{y_i}) \tag{9-23}$$

上面的公式想表达的意义非常简单（对于数据分为$k > 2$类的情况，结论类似）。

- 对于参数θ_l（被预测值y的概率分布），它等于各类别在训练数据中的占比。
- 对于参数$\boldsymbol{\mu}_l$（各类别的中心位置），它等于训练数据里各个类别的平均值（自变量$\boldsymbol{X}_i$的平均值）。
- 对于矩阵参数$\boldsymbol{\Sigma}$（各变量的协方差），它等于各个类别内部协方差的加权平均，权重为类别内数据的个数。

[19] 完整的实现请参考随书配套的代码/ch09-generative_models/da/multivariate_normal.py。

9.3.2 线性判别分析与逻辑回归比较[20]

有了线性判别分析的模型假设（公式（9-22）），我们现在来看看这个模型的预测公式。根据贝叶斯框架里的公式（9-6），线性判别分析的预测公式如下：

$$P(y = 1 \mid \boldsymbol{X}) = \frac{P(\boldsymbol{X} \mid y = 1)P(y = 1)}{P(\boldsymbol{X} \mid y = 0)P(y = 0) + P(\boldsymbol{X} \mid y = 1)P(y = 1)} \tag{9-24}$$

经过一系列数学运算后[21]，可以得到：

$$P(y = 1 \mid \boldsymbol{X}) = {}^{1}/_{(1 + \mathrm{e}^{-\boldsymbol{X}\boldsymbol{\beta}})} \tag{9-25}$$

这就是逻辑回归的预测公式（细节请参考 5.1.3 节），也就是说从预测的角度来看，线性判别分析和逻辑回归是一模一样的。那这两个模型有什么差别呢？

线性判别分析的模型假设更强一点，它首先假设在已知数据类别（因变量）的条件下数据的自变量是正态分布，再通过贝叶斯定理得到被预测值（因变量）的概率分布。当数据的自变量明显不可能是正态分布时，比如数据里有人的年龄这个自变量，线性判别分析的效果一定不好，因为数据并不满足模型假设。而逻辑回归通常能很好地处理这种数据，因为它并没假设自变量是条件正态分布的。当然，如果数据的自变量真的满足线性判别分析的假设，数学上可以证明，这种情况下，线性判别分析是更加高效[22]的模型。

事实上，数学上可以证明，若假设$\boldsymbol{X} \mid y = l \sim Poisson(\lambda_l)$，即自变量的条件分布是泊松分布（Poisson distribution）时，则模型的预测公式$P(y = 1 \mid \boldsymbol{X})$也和逻辑回归一样。我们可以进一步证明，还很多类型的概率分布（自变量服从的先验分布）能推导出逻辑回归的预测公式。这从数学上证明了，逻辑回归其实包括了很多先验分布，正态分布只是其中的一种，如图 9-14 所示。

从线性判别分析和逻辑回归这个例子上可以看到，相比于判别式模型，生成式模型是“小而精”的模型。因为它们的模型假设更强，往往适用场景不多，但一旦问题场景满足特定的假设，生成式模型的预测效果往往会更好。

---

[20] 本节的内容主要参考 Andrew Ng 在斯坦福大学的公开课《CS 229 Machine Learning》。

[21] 这里给出计算的大致步骤。对公式（9-24）的分子分母同时除以$P(\boldsymbol{X} \mid y = 1)P(y = 1)$，可以得到：

$$P(y = 1 \mid \boldsymbol{X}) = {}^{1}/_{[1 + P(\boldsymbol{X} \mid y = 0)P(y = 0)/P(\boldsymbol{X} \mid y = 1)\,P(y = 1)]}$$

因此只需证明$P(\boldsymbol{X} \mid y = 0)/P(\boldsymbol{X} \mid y = 1)$能写成$\mathrm{e}^{-\boldsymbol{X}\boldsymbol{\beta}}$的形式。为了推导方便，将正态分布的概率分布函数记为$f(\boldsymbol{X}) = A\mathrm{e}^{-0.5(\boldsymbol{X}-\boldsymbol{\mu})\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu})^{\mathrm{T}}}$。又因为$(\boldsymbol{X}-\boldsymbol{\mu_0})\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu_0}) - (\boldsymbol{X}-\boldsymbol{\mu_1})\boldsymbol{\Sigma}^{-1}(\boldsymbol{X}-\boldsymbol{\mu_1}) = \boldsymbol{X}\boldsymbol{\alpha} + \gamma$，其中，$\boldsymbol{\alpha}, \gamma$的具体表达式并不重要。在原来的自变量基础上，加入常数项，仍记为$\boldsymbol{X}$，则可以得到：

$$P(\boldsymbol{X} \mid y = 0)P(y = 0)/P(\boldsymbol{X} \mid y = 1)\,P(y = 1) = \mathrm{e}^{-\boldsymbol{X}\boldsymbol{\beta}}$$

[22] 用学术一点的语言表达就是，线性判别分析是渐进有效的（asymptotically efficient），可以理解为针对正文中的问题，在大数据量下，它是最好的估计。

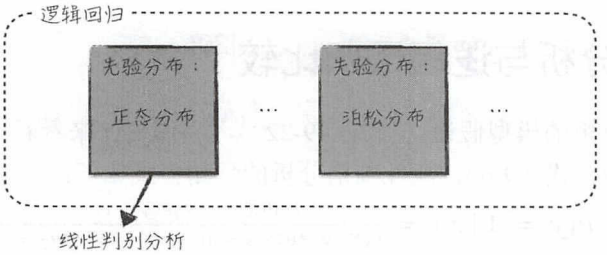


图 9-14

稍稍离题一下，从公式（9-25）中可以得知，$P(y = 1 \mid \boldsymbol{X}) = 0.5$对应着$\boldsymbol{X\beta} = 0$。从空间直观上，这表示，由模型决定的类别分离面（决策边界，decision boundary）是空间中的超平面，比如在二维空间中的一条直线。这就是线性判别分析名字中“线性”的来源：模型的决策边界是线性的。

9.3.3 数据降维

在实际应用中，线性判别分析除了被用于解决分类问题外，还常被用于数据降维。

数据降维（dimension reduction）指的是将高维空间里的数据映射到低维度空间（与之前介绍的核函数刚好相反）。这样的操作虽然不可避免地会损失部分数据信息，但却是机器学习中很常用的技巧。归纳起来，将数据降维通常有 3 个目的。

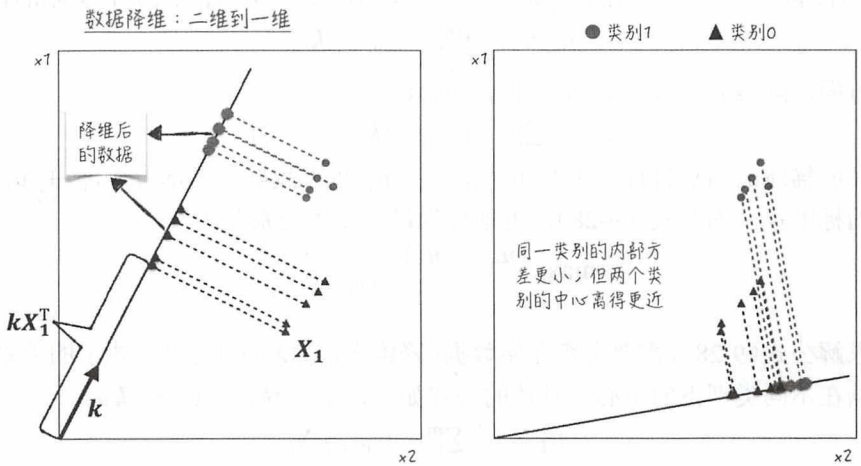
- 除去随机因素的干扰。在实际生产中收集到的数据往往包含了许多随机的噪声项，比如测量时的误差等。这些噪声项不仅对预测结果完全没有帮助，而且还会干扰正常变量对被预测量的影响，从而降低模型的预测效果。数据降维由于是从高维数据降低到低维空间，因而常常能过滤掉这些噪声干扰。
- 避免过拟合问题。数据的维度越大意味着模型参数的个数也就越多，这很容易引起过拟合的问题，特别是在数据量不够大的时候。而数据降维能将多个变量“浓缩”成数量较少的几个变量，这能帮助我们抓住数据变化的主要因素，并舍弃那些不太重要的次要因素，从而有效低避免过拟合的问题。
- 降低工程实现上的难度。对于绝大多数模型，我们都没办法得到它模型参数的解析表达式，需要用特定的最优化算法求解，比如第 6 章中介绍的随机梯度下降法。这些算法需要耗费大量的时间才可能得到收敛的估计值。特别是在模型参数过多时，往往无法在可控的时间内，得到有效的参数估计，导致模型的实用价值降低。而数据降维能有效地解决这个问题。

但之前讨论的线性判别分析跟这 3 点都没有什么联系，而且在搭建模型时，并没有涉及任何的空间变换。那么线性判别分析是如何做到数据降维的呢？

为了回答这个问题，需要了解在数据有标签（数据分为不同类别）的情况下，降维本身是

书籍是人类进步的阶梯

如何进行的。为了表述简洁，同 9.3.1 节类似，假设数据分为两类（对于数据分为$k > 2$类的情况，讨论过程类似，只是数学推导更为复杂），记为$y = 0$或者$y = 1$，而且只有两个自变量，记为$\boldsymbol{X} = (x_1, x_2)$，数据的分布如图 9-15 所示，其中圆点表示类别 1，三角形表示类别 0。



（a）　　（b）

图 9-15

由于在二维空间中，任何一维子空间在直观上都表现为空间中的一条直线。因此将二维数据降为一维数据，从直观上就是让二维空间中的点向某条直线做垂线，而这些垂线与直线的交点（点在直线上的投影）就是降维后的数据，如图 9-15a 中的标记所示[23]。

对数据降维的方法有很多，事实上二维空间中的任意一条直线都对应着一种降维方法，比如图 9-15b 就表示另一种降维结果。那么哪种降维方式更好呢？主要从如下的两个方面来考虑这个问题。

- 由于数据分为不同的两类，我们希望降维之后的数据能尽可能地保留类别之间的差异。也就是说，降维后各类别的中心离得越远越好。
- 同时，对于一个类别里的数据，我们希望通过降维将它们尽可能地聚集到一起。也就是说，降维后各类别内部的方差越小越好。

上面的这两个目标往往是相互矛盾的，比如相比于图 9-15a，图 9-15b 中两个类别的内部方差更小，但两个类别的中心离得更近。为了更加量化地平衡这两方面，需要借助更严谨

[23] 对数学不太熟悉的读者可能会比较奇怪，垂线与直线的交点仍然是二维空间中的点，为什么说它们就是一维数据呢？这是因为在向量空间里，数据的维度其实指的是对应线性空间的维度，后者并不等于向量的长度（向量的长度决定了向量空间维度的上界，比如在这个例子中，向量的长度等于 2，因此数据最多是二维的）。垂线与直线的交点显然都在一条直线上，也就是说用一个向量就能线性表示所有数据，因此这些点都在一维线性空间里，表示的数据也是一维的。

书籍是人类进步的阶梯

的数学工具。

首先将上述的降维过程用数学语言描述出来，如图 9-15a 所示，假设直线的向量[24]为$\boldsymbol{k}$，且$\|\boldsymbol{k}\| = 1$，则点X_1在这条直线上的投影长度（投影到原点的距离）可以表示为$\boldsymbol{k}\boldsymbol{X}_1^{\mathrm{T}}$。一方面，在降维后，两个类别的中心$u_0, u_1$如公式（9-26）所示，其中，$N_0, N_1$为各类别的数据个数。

$$u_l = \frac{1}{N_l}\sum_{i=1}^{m} 1_{\{y_i = l\}}\boldsymbol{k}\boldsymbol{X}_i^{\mathrm{T}} \tag{9-26}$$

另一方面，降维后，每个类别内部方差v_0, v_1为：

$$v_l = \sum_{i=1}^{m} 1_{\{y_i = l\}}(\boldsymbol{k}\boldsymbol{X}_i^{\mathrm{T}} - u_l)^2 \tag{9-27}$$

结合上面描述的两点目标：中心距离$|u_0 - u_1|$越大越好，内部方差$v_0 + v_1$越小越好，数据降维的标准定义为公式（9-28），可理解为最大化“类别距离”：

$$\max_{\boldsymbol{k}} (u_0 - u_1)^2 \Big/ (v_0 + v_1) \tag{9-28}$$

为了求解公式(9-28)，需要考查原始数据（降维之前）$\{\boldsymbol{X}_i, y_i\}$与这个式子的关系。记$\boldsymbol{\mu}_0, \boldsymbol{\mu}_1$为原始数据在不同类别下的中心，具体的公式如下，很容易证明$u_l = \boldsymbol{k}\boldsymbol{\mu}_l^{\mathrm{T}}$。

$$\boldsymbol{\mu}_l = \frac{1}{N_l}\sum_{i=1}^{m} 1_{\{y_i = l\}}\boldsymbol{X}_i \tag{9-29}$$

在上面式子的基础上，定义相应的类别内部方差矩阵$\boldsymbol{s}_0, \boldsymbol{s}_1$如下：

$$\boldsymbol{s}_l = \sum_{i=1}^{m} 1_{\{y_i = l\}}(\boldsymbol{X}_i - \boldsymbol{\mu}_l)^{\mathrm{T}}(\boldsymbol{X}_i - \boldsymbol{\mu}_l) \tag{9-30}$$

数学上可以证明$v_l = \boldsymbol{k}\boldsymbol{s}_l\boldsymbol{k}^{\mathrm{T}}$，同样地，可以证明$(u_0 - u_1)^2 = \boldsymbol{k}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)\boldsymbol{k}^{\mathrm{T}}$。不妨记$\boldsymbol{s}_W = (\boldsymbol{s}_0 + \boldsymbol{s}_1)$以及$\boldsymbol{s}_B = (\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)^{\mathrm{T}}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$。在学术上，$\boldsymbol{s}_W$被称为 within-class scatter matrix，而$\boldsymbol{s}_B$被称为 between-class scatter matrix。有了这些定义，可以用它们重写数据降维的标准（公式（9-30）），具体的如公式（9-31）所示：

$$\max_{\boldsymbol{k}} \boldsymbol{k}\boldsymbol{s}_B\boldsymbol{k}^{\mathrm{T}} \Big/ \boldsymbol{k}\boldsymbol{s}_W\boldsymbol{k}^{\mathrm{T}} \tag{9-31}$$

略去一些繁琐的计算[25]，可以得到降维的最佳选择$\boldsymbol{k}^* = \boldsymbol{s}_W^{-1}(\boldsymbol{\mu}_0 - \boldsymbol{\mu}_1)$。值得注意的是，$\boldsymbol{\mu}_l$与公式（9-23）中的$\widehat{\boldsymbol{\mu}}_l$（线性判别分析里中心位置的估计值）是一样的，以及$\frac{1}{m}\boldsymbol{s}_W$与公式（9-23）中的$\widehat{\boldsymbol{\Sigma}}$（线性判别分析里协方差矩阵的估计值）是一样的。也就是说，线性判别分析与上述的数据降维过程其实是等价的，前者的估算答案决定了后者的最优解，如图 9-16 所示。因此，不妨将上面的降维过程称为线性判别分析降维。

从表面上来看，线性判别分析似乎对所有分类数据都能进行降维，因为上面的推导过程

---

[24] 此时直线的方程为$\boldsymbol{w}\boldsymbol{X}^{\mathrm{T}} = 0$，其中向量$\boldsymbol{w}$和向量$\boldsymbol{k}$垂直。

[25] 数学上可以证明，公式（9-31）等价于求解下面的矩阵方程，因此，$\boldsymbol{k}$其实是矩阵$\boldsymbol{s}_W^{-1}\boldsymbol{s}_B$的特征向量（对应着最大的特征值）。

$$\boldsymbol{s}_W^{-1}\boldsymbol{s}_B\boldsymbol{k}^{\mathrm{T}} = \lambda\boldsymbol{k}^{\mathrm{T}}$$

书籍是人类进步的阶梯

并没有对数据做任何假设。但事实上，降维与解决分类问题是等价的，因此，当自变量明显不符合正态分布时，线性判别分析降维的效果将无从保证：经过降维后，不同类别的数据将混杂在一起，不再保留类别间的差异。

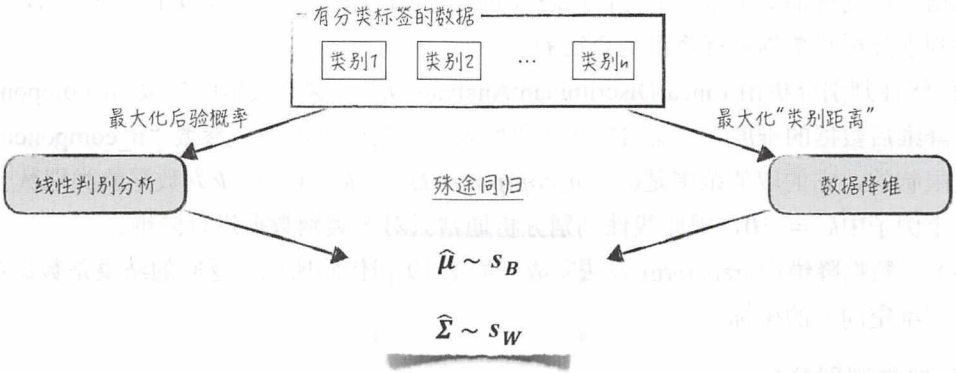


图 9-16

在机器学习领域，除了这里讨论的线性判别分析，还有很多其他的数据降维算法，比如常用的主成分分析（PCA），具体的细节将在第 10 章中讨论。

9.3.4 代码实现

为了展示模型效果，我们将使用 scikit-learn 自带的数据集——手写数字图像。数据集里的数据被分为 10 类，分别对应 0～9 这 10 个数字。数据有 64 个自变量，分别表示每个像素点的亮度（也就是说，数字图像的分辨率为8 × 8），具体的如图 9-17 所示。

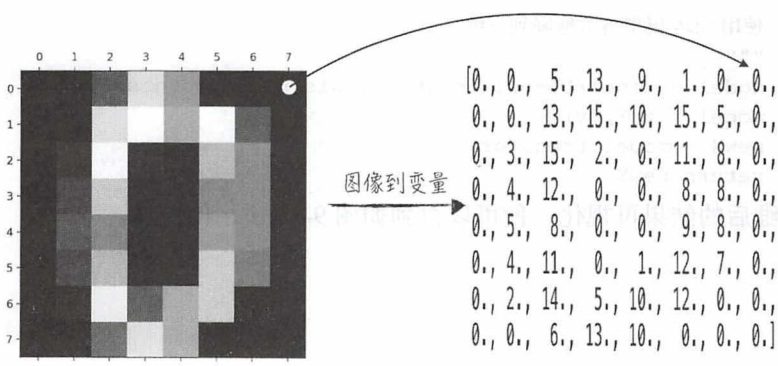


图 9-17

书籍是人类进步的阶梯

模型的代码实现如程序清单 9-5 所示。

（1）通过 datasets.load_digits 方法读取数据，如第 8 行代码所示。数据的自变量存储在“data”字段，而数据的类别存储在“target”字段，如第 9、10 行代码所示。值得注意的是，虽然数据的自变量都是整数，不符合正态分布，但它们的先验分布近似于正态分布，因此使用线性判别分析对数据进行降维是合适的。

（2）线性判别分析由 LinearDiscriminantAnalysis 实现，这个类里的参数“n_components”决定了降维后数据的维度，如第 17 行代码所示。需要注意的是，参数“n_components”是有一定限制的，它的取值范围是$0 < n_components < k$。其中，k为数据的类别数[26]，比如在这个例子中$k = 10$，因此线性判别分析通常只对多类别数据进行降维。

（3）对数据降维由 transform 方法完成，如第 19 行代码所示，返回的结果是数据在新空间里（三维空间）的坐标。

程序清单 9-5 线性判别分析

```
1 | from sklearn import datasets
2 | from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
3 |
4 | def loadData():
5 |     """
6 |     读取 scikit-learn 自带数据：手写数字
7 |     """
8 |     digits = datasets.load_digits()
9 |     X = digits.data
10 |    y = digits.target
11 |    return X, y
12 |
13 | def dimensionReduction(X, y):
14 |     """
15 |     使用 LDA 模型将数据降到三维
16 |     """
17 |     model = LinearDiscriminantAnalysis(n_components=3)
18 |     model.fit(X, y)
19 |     newX = model.transform(X)
20 |     return newX
```

将数据降维后的结果可视化，便可以得到如图 9-18 所示的图像。

[26] 数学上可以证明，在公式（9-31）中，矩阵s_B的秩最多为$k - 1$。因此降维后，数据的维度最多为$k - 1$。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

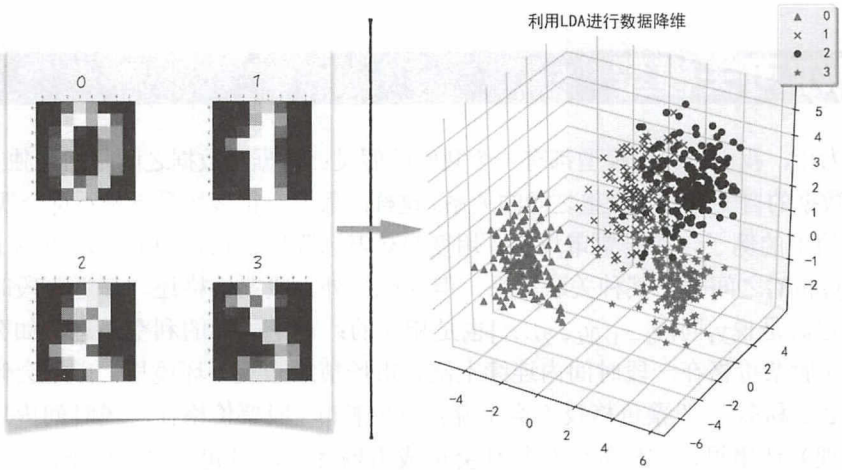


图 9-18[27]

9.3.5 二次判别分析

在线性判别分析中，模型假设在不同类别下，自变量分布的协方差矩阵是一样的，而二次判别分析（Quadratic Discriminant Analysis，QDA）放松了这个假设。以二元分类问题为例，二次判别分析的假设如公式（9-32）所示：

$$\begin{aligned} \boldsymbol{X} \mid y = 0 &\sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \\ \boldsymbol{X} \mid y = 1 &\sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \\ P(y = 0) = \theta_0, P(y = 1) &= \theta_1 \end{aligned} \tag{9-32}$$

正如 9.3.1 节中讨论的，正态分布的协方差代表了分布的形状。因此，二次判别分析可以理解为区分类别的中心位置，也区分不同类别的在形状上的差异。

当公式（9-32）中的协方差矩阵都为对角线矩阵时，二次判别分析就变成了朴素贝叶斯里的高斯模型（Gaussian naive Bayes）。事实上，线性判别分析也可以被视为二次判别分析的一个特例，将这个特例单独命名研究的目的就在于线性判别分析可以用于数据降维，而一般二次判别分析则没有这个功能。

对于高斯模型和二次判别分析，在第三方库 scikit-learn 中，它们相应的实现为 GaussianNB 以及 QuadraticDiscriminantAnalysis。具体的调用方式与其他模型类似，在此不再赘述。

[27] 完整的实现请参考随书配套的代码/ch09-generative_models/da/dimension_reduction.py。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

9.4 隐马尔可夫模型

到目前为止，我们讨论的模型都有一个共同的假设：数据与数据之间是相互独立的，模型只关注当前数据的自变量与因变量之间的关系。这样的假设在很多场景下是不符合现实情况的。

举一个简单的例子，在股票市场上，用变量y_i表示股票价格，用$\boldsymbol{X_i}$表示市场上的公开信息。显然，y_i和$\boldsymbol{X_i}$之间存在某种关联关系。但除此之外，股票价格还与最近一段时间的市场环境相关，也就是说y_i同$\{y_{i-1}, y_{i-2}, \ldots\}$也是相关的：对于同样的利空消息，如公司出现亏损，在牛市（股票价格在一段时间内连续上涨，市场情绪乐观）环境里，市场会倾向认为这是“利空出尽是利多”，股票价格反而会上涨；但在熊市（股票价格在一段时间内连续下跌，市场情绪悲观）环境里，这样的消息往往会造成市场恐慌，引起股票价格下跌。

类似的例子在现实生活中还有很多，它们的共同点是：数据间的顺序对数据本身是有影响的。在学术上，具有这种特性的数据被称为序列数据（sequential data）。显然之前讨论的模型都没办法捕捉到数据中的“顺序”特性，而本节将讨论的隐马尔可夫模型（Hidden Markov Model， HMM）就是为解决这个问题而设计的。正因为这个优点，它在基因分析、语言识别以及量化金融等领域被大量使用。

需要提醒读者注意的是，隐马尔可夫模型这个名字与朴素贝叶斯类似，指的是一类模型，而非单个模型。

9.4.1 一个简单的例子

隐马尔可夫是一个比较复杂的模型，数学的处理也很繁琐。为了便于理解，我们先来看一个简单的例子[28]。

数据科学家小安有 3 天假期，她将根据天气情况来决定这 3 天的活动。天气情况分为两种：下雨或者晴天；而小安的备选活动有 3 个：郊游、逛街和在家打游戏。天气情况的假设如下：

- 每天的天气情况与前一天的天气是相互关联的；
- 如果前一天是下雨，则第二天有 70%的概率仍然下雨，有 30%的概率第二天将转为晴天；
- 如果前一天是晴天，则有 60%的概率第二天仍是晴天，有 40%的概率第二天将下雨；
- 另外第一天下雨的概率为 60%，晴天的概率为 40%。

小安将按如下的假设选择她的活动：

[28] 本例参考自维基百科。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

- 在下雨天，3 项活动的概率分别为 10%、40%以及 50%；
- 而在晴天，3 项活动的概率分别为 60%、30%以及 10%。

用图形表示上述的假设可以得到图 9-19。

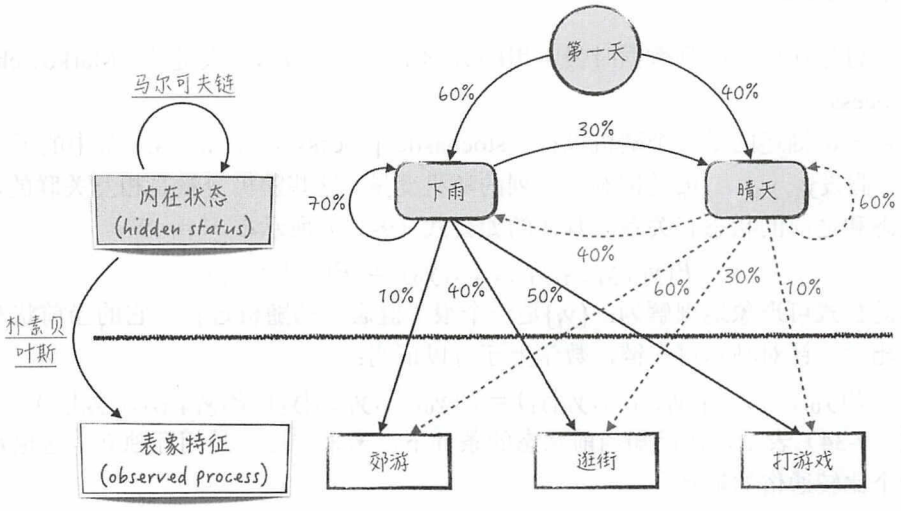


图 9-19

现在观察到小安在 3 天假期内的活动按顺序分别是郊游、逛街和打游戏，需要据此推测这 3 天假期的天气情况，这就是隐马尔可夫模型的典型应用场景。现在将这个例子抽象，用变量y_i表示第i天的天气，是需要被预测的量；x_i表示第i天的活动，是被观测到的量，则模型的假设可以分解为如下两方面。

- 对于单独的某一天来讲，y_i与x_i之间的关系由简单的生成式模型处理，更具体一点，通常使用朴素贝叶斯模型捕捉它们之间的关系，比如这个例子使用的模型就是朴素贝叶斯中的多项式模型（细节可参考 9.2.3 节）。
- 对于被预测量y_i，它们之间也是相互关联的，并且假设当前状态只与前一个状态相关，即y_i只与y_{i-1}相关。这样的关系在数学上被称为马尔可夫链，这也是模型名字的来源，具体的细节将在 9.4.2 节中讨论。

稍微扩展一下，从图形直观上来看，隐马尔可夫模型是一个基于贝叶斯框架的网络。事实上，很多复杂的模型都可以表示成网络的形式，这些模型在学术上它常被形象地称为图模型[29]（graphical model）。图模型无论是在理论上还是在工程实现上都是十分复杂的，是机器学习里非常前沿的领域，而隐马尔可夫是一种很简单的（可能是最简单的）图模型。

[29] 图模型并不是指专门处理图形的模型，而是指模型的架构可以表示为一张网状图。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

从模型的架构上来讲，隐马尔可夫又能被看作多个朴素贝叶斯模型的联结。这再次说明模型联结主义（connectionism）的强大。

9.4.2 马尔可夫链

本节将讨论在处理序列数据时最常用的数学工具——马尔可夫链[30]（Markov chain 或者 Markov process）。

马尔可夫链描述的是一个随机过程（stochastic process），比如 9.4.1 节中的天气情况。更一般地，假设$y_1, y_2, \ldots, y_n$是按顺序排列的随机变量。这些随机变量是相互关联的，也就是说当前状态和之前的状态有关系，具体的如公式（9-33）所示：

$$P(y_i \mid y_{i-1}, y_{i-2}, \ldots, y_0) = P(y_i \mid y_{i-1}) \tag{9-33}$$

上面的公式可形象地理解为：$\{y_i\}$是一个很“健忘”的随机过程，它的当前状态只跟前一个状态相关。针对马尔可夫链，数学上还可以证明：

$$P(y_0, \ldots, y_{i-1}, y_{i+1}, \ldots y_n | y_i) = P(y_0, \ldots, y_{i-1} | y_i) \cdot P(y_{i+1}, \ldots, y_n | y_i) \tag{9-34}$$

公式（9-34）表示，在已知当前状态的条件下，未来和过去是相互独立。这也是马尔可夫链另一个比较通俗的解释。

当$\{y_i\}$的取值离散时，相应的马尔可夫链[31]可由转移矩阵（transition matrix）和初始分布（即随机变量y_0的分布）表示。比如 9.4.1 节中的天气情况可由一个2×2的矩阵和相应的初始分布表示，如图 9-20 所示。

更一般地，假设y_i可能的取值有n个，则转移矩阵$\boldsymbol{Q}$为$n \times n$的方阵，矩阵中元素的表示式为：

$$\boldsymbol{Q}_{i,j} = P(y_i = j \mid y_{i-1} = i) \tag{9-35}$$

也就是说，转移矩阵中第i行第j列的元素为状态i到状态j的概率。因此第i行表示状态i到其他可能状态的概率分布，显然它们的和等于 1，即$\sum_{j=1}^{n} \boldsymbol{Q}_{i,j} = 1$。

以上就是马尔可夫链的数学定义。需要提醒读者注意的是，虽然它定义当前状态只跟前一状态相关，如公式（9-33）所示，但其实稍加变换就可以处理更为复杂的情况。比如假设当前状态跟最近的两个状态都相关，即$P(y_i \mid y_{i-1}, y_{i-2}, \ldots, y_0) = P(y_i \mid y_{i-1}, y_{i-2})$。针对这种情况，定义新的状态$\boldsymbol{z}_i = (y_i, y_{i-1})$，显然$\{y_i\}$和$\{\boldsymbol{z}_i\}$是等价的，而且容易证明随机过程$\boldsymbol{z}_i$是一个马尔可夫链。

---

[30] 马尔可夫链以俄国数学家安德雷·安德耶维齐·马尔可夫（Andrey Andreyevich Markov）的名字命名。除了机器学习，这个数学工具还被广泛应用于金融领域，特别是衍生品市场。

[31] 数学上严谨的表达是：针对离散情况，一个平稳马尔可夫链（stationary Markov chain）可以定义相应的转移矩阵。马尔可夫链被称为平稳的，当且仅当$P(y_{i+1} = k \mid y_i = l) = P(y_i = k \mid y_{i-1} = l)$，也就是说状态间的转换概率不随时间变化。本章讨论离散情况下的隐马尔可夫模型，它使用的就是平稳马尔可夫链。事实上，在人工智能领域，几乎不会用到非平稳的马尔可夫链。

书籍是人类进步的阶梯

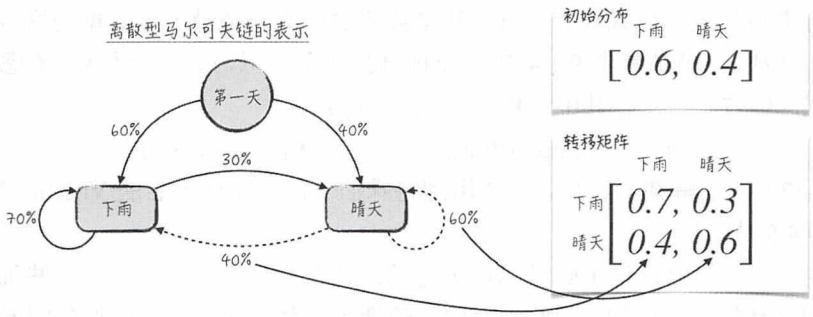


图 9-20

9.4.3 模型架构

借助上面讨论的马尔可夫链，离散情况下的隐马尔可夫的模型参数（限于篇幅，本章只讨论y_i是离散的情况）可以被分解如下 3 个部分。

- 先验概率$P(\boldsymbol{X_i} \mid y_i)$：这部分模型与之前讨论的生成式模型没有任何差别。事实上，在实际生产中，常用朴素贝叶斯模型来处理$P(\boldsymbol{X_i} \mid y_i)$。
- 初始分布$P(y_0)$：在离散的情况下，它就是一个多项式分布。
- 转移矩阵$P(y_i \mid y_{i-1})$：它与上面的初始分布一起组成内在状态（hidden status）的马尔可夫链。

根据上面的分析，可以将隐马尔可夫模型表示成如图 9-21 所示的图像。

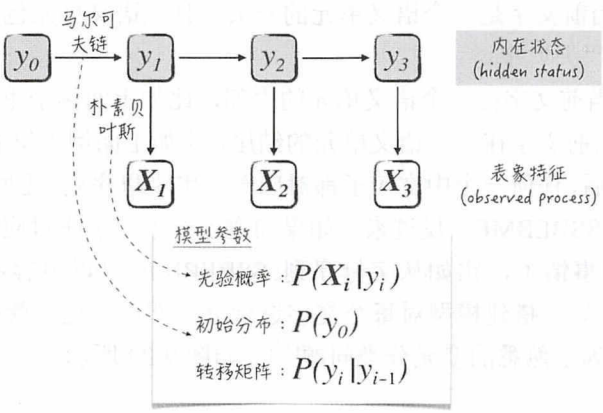


图 9-21

根据上面假设的模型参数，可以得到数据的联合概率：

$$P(\boldsymbol{X}, y) = P(y_0)P(y_1 \mid y_0)\ldots P(y_i \mid y_{i-1})\prod_j P(\boldsymbol{X_j} \mid y_j) \tag{9-36}$$

书籍是人类进步的阶梯

隐马尔可夫与其他生成式模型一样，其参数的估计原则是最大化数据的联合概率（也称为最大似然估计法，细节请参考 9.1.4 节）；而预测公式也类似，只是需要整体考虑所有数据，具体地如公式（9-37）所示，其中，$\boldsymbol{Y} = (y_0, \ldots, y_T)$。

$$\hat{\boldsymbol{Y}} = \operatorname{argmax}_{\boldsymbol{Y}} P(y_0, y_1, \ldots, y_T \mid \boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_T) \tag{9-37}$$

公式（9-37）的求解并不容易，需要用到特殊的算法，比如著名的 Viterbi 算法，具体的细节请参考 9.4.5 节。

到目前为止，我们只讨论了隐马尔可夫链的理论部分。这些内容显得有些抽象，不太容易理解模型到底是如何运行的。因此下面将讨论两个具体的例子，它们将分别展示如何在监督式学习和非监督式学习领域使用隐马尔可夫模型。

9.4.4 中文分词：监督式学习

9.2.5 节中讨论中文处理时曾提到，分词是其中很重要的一步，也是隐马尔可夫模型很典型的应用场景（监督式学习）。下面就来详细讨论中文分词的详细步骤。

所谓中文分词就是将句子里的一个或多个文字组合在一起构成中文里面的语义单元，也就是日常生活中词的概念。比如“我爱北京天安门”这句话就分为 4 个语义单元，分布是“我”“爱”“北京”和“天安门”。

为了能到达上面的分词效果，我们首先需要将其转换为模型能处理的分类问题。具体地，假设文字一共有 4 种状态，分别以字母 B、M、E、S 表示。

- 字母 S 表示当前文字单独成一个语义单元。比如上面例子里的“我”和“爱”字。
- 字母 B 表示当前文字是一个语义单元的开头，且该语义单元包含多个文字。比如上面例子里的“北”和“天”字。
- 字母 M 表示当前文字在一个语义单元的内部，比如上面例子里的“安”字。
- 字母 E 表示当前文字在一个语义单元的结尾，比如上面例子里的“京”和“门”字。

经过这样的定义后，任何一个中文句子都对应着一串字母序列，比如“我爱北京天安门”这句话就对应着序列 SSBEBME。反过来，如果知道了一个句子所对应的字母序列，那么分词就是一件很简单的事情了，比如从字母序列 SSBEBME 可以很容易得到分词的结果：S/S/BE/BME。因此，只需搭建模型对每个文字进行分类即可。这样就把中文分词这件看似无从下手的问题转换为了熟悉的多元分类问题[32]，如图 9-22 所示。

---

[32] 在实际生产中，对中文分词并不会只将文字状态分为 4 类，因为这样的分类方法太“粗糙”，并不足以解构复杂的中文，导致模型效果并不好。在实际中，通常会将分词和词性标注（名词、动词等）结合在一起。比如在正文的例子里，用字母 V 表示动词，则“爱”字的状态为“S+V”。经过这样的转换后，虽然表面上增加了模型的预测难度（被预测量的类别更多），但实际上，分词效果会明显上升。

当然这两种分类方法对模型而言是没有任何差别的，因此为了表述简洁，在正文里只讨论较为直接的第一种方法。

书籍是人类进步的阶梯

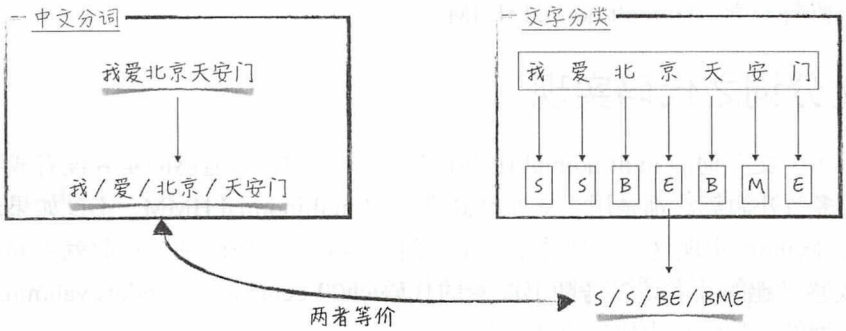


图 9-22

显然在这个问题中，文字之间不是相互独立的，比如如果一个字的状态是 B，那么紧接着它的文字不可能是状态 S，于是使用隐马尔可夫模型来解决这个分类问题。

具体地，用变量y_i表示第i个字的状态，根据上面的假设y_i可能的取值有 4 个；用变量x_i表示第i个字具体是什么文字。根据 9.4.3 节中的讨论，现搭建模型如下。

- 先验概率$P(x_i \mid y_i)$：在已知文字状态的情况下，每个汉字出现的概率。针对这部分概率，使用多项式模型（包括多项式模型里的特征提取方法，细节请参考 9.2.3 节），假设$P(x_i = k \mid y_i = l) = p_{k,l}$。
- 初始分布$P(y_0)$：表示每句话第一个字的状态分布，这个比较直接，假设$P(y_0 = l) = a_l$。
- 转移矩阵$P(y_i \mid y_{i-1})$：表示 4 种状态间相互转移的可能性，这部分也比较直接，假设矩阵的元素为$P(y_i = l \mid y_{i-1} = t) = b_{l,t}$。

在中文分词这个问题中，我们能够观察到被预测量y_i（在实际应用里，字的状态来源于人工标注）。因此根据最大似然估计法的原则（公式（9-36）以及 9.1.4 节），容易得到上述参数估计值的解析表达式。

$$\hat{p}_{k,l} = C(x = k \,\&\, y = l) \Big/ C(y = l)$$

$$\hat{a}_l = C(y_1 = l) \Big/ C(y_1)$$

$$\hat{b}_{l,t} = \sum_i C(y_i = l \,\&\, y_{i-1} = t) \Big/ \sum_i C(y_{i-1} = t) \tag{9-38}$$

公式(9-38)中C表示事件在训练数据中出现的次数，比如$C(y = B)$表示在训练文本中，状态 B 出现的次数（不管它的具体位置是什么）。由此可见，这个隐马尔可夫模型的参数估计值与多项式模型非常类似，都是具体事件在训练数据里的占比[33]，因此在学术上这个模型

[33] 与多项式模型类似，在实际应用中，常在公式（9-38）的基础上加入平滑项，由平滑系数α控制。具体的方法与公式（9-18）类似。

书籍是人类进步的阶梯

被称为用于监督式学习的 multinomial HMM。

9.4.5 中文分词之代码实现

虽然用于中文分词的 multinomial HMM 模型比较简单，但遗憾的是并没有成熟的开源解决方案，大多数开源算法都是用于非监督式学习的 multinomial HMM。不过如果理解了模型的原理，用 Python 实现这个模型并不困难。限于篇幅，代码的细节在此就不展开了（对工程实现比较感兴趣的读者请参考随书配套的代码/ch09-generative_models/yahmm/hmm[34]），仅讨论在模型的工程实现中比较重要的几点。

1. 单元测试

就程序本身而来，模型的实现并不复杂。通常只需要几百或者几千行代码，但与普通的程序相比，模型本身包含了比较复杂的数学逻辑，很难保证编写的代码就是想要的模型逻辑。因此单元测试就显得极其重要了，否则很容易陷入这样的困境——“当模型效果不好时，不知道是因为自己的模型搭得不好，还是因为代码写错了”[35]。而且做单元测试时，不能只简单地测试功能是否实现，更需要用人工计算的模型结果去验证代码的逻辑是否有问题。

对代码做单元测试通常会用到一些自动化的辅助工具，在 Python 中，nosetests 是一个不错的选择。

2. Cython

Python 虽然很简单易学，但它最大的问题是运行速度太慢，用 Python 来做机器学习的模型运算几乎不可行。为了解决这个问题，比较成熟的第三方库通常用 C 语言实现具体的复杂运算，并将其封装好，供 Python 调用（之前章节一直使用的 scikit-learn 就是这样的实现架构，只是这些细节对用户是不可见的，所以容易造成是 Python 在计算模型的假象）。

正因为如此，自己实现算法时也需要用 C 语言来封装模型的核心算法，比如隐马尔可夫模型中的 Viterbi 算法。但问题是 C 语言比较复杂，实现起来难度较大，有没有比较简单的实现方法呢？答案是使用 Cython，它能将大部分 Python 代码自动地“翻译”成 C 代码，并将后者封装好，供 Python 调用。

在随书配套的代码中，用 Python 和 Cython 分别实现了 Viterbi 算法，它们的代码路径分别是 hmm/utils/viterbipy.py 和 hmm/utils/viterbi.pyx。如果比较两个脚本会发现，绝大部分代码都是一样的，但经过编译[36]后，Cython 版的算法比 Python 版的快 10～100 倍。有关 Cython

---

[34] 这个模型 API 设计参考自 GitHub 上的 larsmans/seqlearn。虽然后者是 scikit-learn 的相关项目，但其在实现上有很严重且致命的 bug，而且无人维护，不推荐读者使用。

[35] 引自美国伯克利加州大学 Michael Irwin Jordan 教授的观点。

[36] Cython 的安装命令为“pip install cython”。编译 Cython 代码的命令为“python setup.py build_ext --inplace”，其中 setup.py 是包含编译命令的 Python 脚本，它的路径为/ch09-generative_models/yahmm/setup.py。

书籍是人类进步的阶梯

的具体细节超出了本书的范围，请读者参考其他相关资料。

回到隐马尔可夫模型，讨论上面多次提到的 Viterbi 算法[37]，该算法负责求解模型预测结果（公式（9-37））。这个算法的思路是利用动态规划（dynamic programming），将最初的问题转换为相似的子问题来解决。具体地，记$S_T(y_T) = \max_{y_0,\ldots,y_{T-1}} P(y_0, y_1, \ldots, y_T \mid \boldsymbol{X_1}, \ldots, \boldsymbol{X_T})$，也就是说$S_T(y_T)$表示针对自变量$\{\boldsymbol{X_1}, \ldots, \boldsymbol{X_T}\}$，在已知第$T$个状态的情况下，所能达到的最大条件概率。数学上可以证明，这个问题可以按如图 9-23 所示的方式将其拆解为子问题，这就是 Viterbi 算法的核心。其中符号$\propto$表示正比于，这样书写是为了避免那些对结果没有影响但又极其繁琐的数学公式，在机器学习领域很常用。至于 Viterbi 算法实现的具体代码，限于篇幅，在此就不做讨论了。

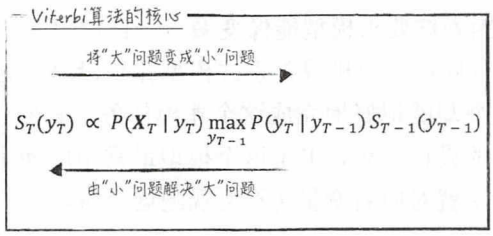


图 9-23

最后，使用实现好的 multinomial HMM 模型对中文进行分词，训练模型的文本来自 GitHub 中的 liwenzhu/corpusZh。这份数据是词性标注数据，并没有标准的分词标记。因此在训练模型前，需要对其做数据转换，具体的可参考/ch09-generative_models/yahmm/segmentation_example.py。模型的效果如图 9-24 所示，对中文分词的准确度在 80%左右。

Multinomial HMM的分词效果

	precision	recall	f1-score	support
B	0.77	0.86	0.82	136521
E	0.78	0.86	0.82	136521
M	0.48	0.50	0.49	22136
S	0.86	0.70	0.77	163983
avg / total	0.79	0.79	0.79	459161

示例：

我爱北京天安门: / 我 / 爱 / 北京 / 天安门

图 9-24

[37] 该算法是由美国工程师安德鲁·詹姆斯·维特比（Andrew James Viterbi）发明的，因此以他的名字为算法命名。维特比不仅是一名杰出的工程师，还是一名成功的商人，他是芯片制造商高通公司的联合创始人。

9.4.6 股票市场：非监督式学习

上面的 9.4.4 节和 9.4.5 节讨论了如何在监督式学习的场景下使用隐马尔可夫模型。在监督式学习（针对序列数据）里，搭建模型的训练数据为$\{(\boldsymbol{X}_i, y_i)\}$，其中$y_i$表示事物的状态，它是能被观察的（训练数据里有这个变量）。但在现实生活中的很多情况下，变量y_i是隐藏的、观测不到的，得到的训练数据里只有$\boldsymbol{X}_i$，如图 9-25 所示。比如 9.4.1 节中的例子：只能观察到小安每天的活动，并不知道每天的天气情况；又比如在股票市场，我们只能看到股票价格、成交量等信息，但无从知道当前股市的状态（分为牛市、熊市和震荡 3 种状态）。这种场景在学术上被称为非监督式学习。

对于这种类型的数据，我们希望也能像之前一样，搭建模型预测变量y_i（根据变量$\boldsymbol{X}_i$）。这是一个很困难的任务，因为这要求模型能像变魔术一样，“无中生有”地变成相应的y_i来。之前讨论的逻辑回归、支持向量学习机等判别式模型完全没办法处理这类型数据，但作为生成式模型的隐马尔可夫模型却能很好地完成这个建模任务。事实上，解决这种非监督式学习问题才是隐马尔可夫模型的设计初衷，也是这个模型最常用的使用方法。从文字上来讲，隐马尔可夫这个名字中的隐字就对应着变量y_i不可观测这个事实。

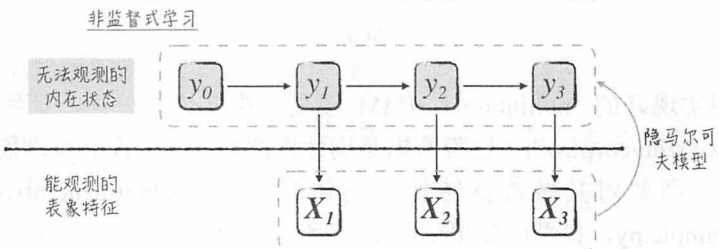


图 9-25

隐马尔可夫模型是如何做到这一点的呢？要回答这个问题，我们先来看看在已知y_i的情况下，隐马尔可夫模型是如何估计模型参数的。正如 9.1.4 节中讨论的，隐马尔可夫模型估计模型参数的原则是最大似然估计法，就是最大化序列数据的联合概率$P(\boldsymbol{X}, y)$，如公式（9-39）所示，其中θ表示模型参数，$(\boldsymbol{X}, y) = \{(X_i, y_i)\}$表示训练数据。

$$P(\boldsymbol{X}, y \mid \theta) = P(y_0 \mid \theta) \prod_i P(y_i \mid y_{i-1}, \theta) \prod_j P(\boldsymbol{X}_j \mid y_j, \theta)$$
$$\hat{\theta} = \operatorname{argmax}_\theta P(\boldsymbol{X}, y \mid \theta) \tag{9-39}$$

在非监督学习中，变量y_i是未知的，但依然可以类似地定义模型的似然函数，如公式（9-40）所示：

$$P(\boldsymbol{X} \mid y, \theta) = P(y_0 \mid \theta) \prod_j P(y_i \mid y_{i-1}, \theta) \prod_j P(\boldsymbol{X}_j \mid y_j, \theta)$$
$$\hat{\theta}, \hat{y} = \operatorname{argmax}_{\theta, y} P(\boldsymbol{X} \mid y, \theta) \tag{9-40}$$

与监督式学习相比（公式（9-39）），变量y_i从已知的训练数据变成了一类特殊的“模型

参数”，但整个似然函数的“架子”（数学公式）还是不变的。针对这类似然函数，依然可以使用最大似然估计法的原则来估计模型参数θ以及被预测量y_i。但与监督式学习相比，由于新加入了未知量y_i，需要估计的“参数”个数更多了，所以需要用到特殊的估计算法：最大期望算法（Expectation-Maximization Algorithm，EM）。下面就以 multinomial HMM 为例介绍，如何在非监督式学习场景里使用 EM 算法估计模型参数。

- 在非监督式学习中，multinomial HMM 的模型假设是不变的。因此，只要给定y_i的取值（先不管这里给定的y_i是否正确合理，事实上y_i可以是随机生成的），那么就能像公式（9-38）那样估算相应的模型参数。
- 反过来，如果有了模型参数（同样地，先不管这些模型参数是否合理），就可以使用 9.4.5 节中讨论的 Viterbi 算法得到相应的预测值y_i。

数学上可以证明，如果将上面的两步重复交叉使用，就可以最终得到公式（9-40）的解。这就是 EM 算法，其中第一步被称为 M step，而第二步被称为 E step，具体如图 9-26 所示。

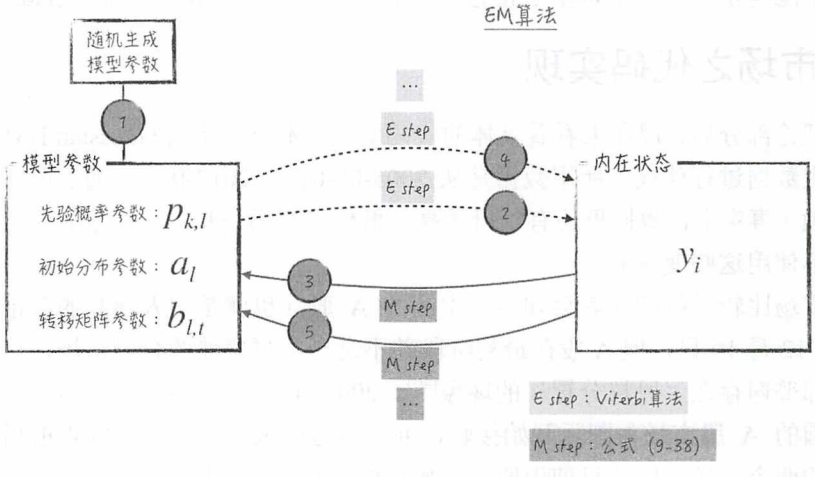


图 9-26

在讨论完有点艰深晦涩的模型理论后，我们重新回到股票市场，讨论如何使用隐马尔可夫模型对其建模。正如本节开头讨论的，将股票市场分为如下两层。

- 第一层是观察得到的股票市场特征包括 5 日的收益率、20 日收益率、5 日成交额增长率以及 20 日成交额增长率，用变量$\boldsymbol{X_i}$表示。
- 第二层是观察不到的市场状态，一共包含 3 种状态，分布是牛市、熊市和震荡（当然也可细分为更多的市场状态，这里仅以 3 种状态举例），用变量y_i表示。

我们希望能搭建这样一个模型，它能根据一段时间的市场表现（变量$\boldsymbol{X_i}$），预测当前的市场状态以及未来一段时间的市场状态（变量y_i）。在金融界，针对股票市场有一个很重要

的假说，那就是有效市场假说[38]。这个假说的意思是当前的股票价格已经反映了所有历史信息的价值。换句话说，当前的市场状态已经完全包含了历史市场状态，因此在已知当前市场的条件下，未来和过去是相互独立的，即y_i可以认为是一个马尔可夫链，这是使用隐马尔可夫模型的基础。除此之外，进一步假设在市场状态已知的情况下，股票的日收益率和成交量服从正态分布[39]。根据这些假设，搭建隐马尔可夫模型如下。

- 先验分布$P(\boldsymbol{X}_i \mid y_i)$：正如上面的假设，这个分布是一个正态分布，而且显然在不同的市场环境下，日收益率和交易量对应的协方差矩阵显然是不一样的。因此，假设$\boldsymbol{X}_i \mid (y_i = l) \sim \mathcal{N}(\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)$（这个假设和 9.3.5 节中讨论的二次线性判别是一致的）
- 初始分布$P(y_0)$：由于变量y_0是离散的，因此假设$P(y_0 = l) = a_l$（这和 9.4.4 节中的 multinomial HMM 是一致的）。
- 转移矩阵$P(y_i \mid y_{i-1})$，它表示 3 种状态间相互转移的可能性，于是假设矩阵的元素为$P(y_i = l \mid y_{i-1} = t) = b_{l,t}$（这和 multinomial HMM 也是一致的）。

由于这个隐马尔可夫模型是基于正态分布的，因此它在学术上被称为 Gaussian HMM。

9.4.7 股票市场之代码实现

讨论完理论部分后，现在来看看具体的代码实现。本节将使用 Gaussian HMM 模型对上证指数的历史数据进行建模。所用数据是从 2005-06-01 到 2017-08-28 的上证指数以及相应的每日成交额（事实上，数据里还有每日最高最低价等有用信息，但为了表示简洁，本节的建模例子并不使用这些变量）。

对金融市场比较了解的读者会知道，中国的 A 股（也就是被人熟知的深市和沪市）开始于 1990 年 12 月 19 日，但 A 股在最初阶段并不是一个很成熟的金融市场，这主要表现在上市公司内部普遍存在“股权分置”的现象[40]。2005 年 6 月开始的股改是一个标志性的事件，表明中国的 A 股市场与国际开始接轨，并逐步迈向成熟。这个时间点前后的股票市场是截然不同的两个市场，因此只使用股改开始后的数据进行建模。

---

[38] 有效市场假说（efficient-market hypothesis）是金融学里最重要的理论之一。这个理论认为，投资者无法通过建模等技术手段从市场上获取超过平均水平的收益，也就是说对股票市场进行建模是无意义的。

[39] 金融学里常常假设市场是一个服从正态分布的随机游走，而且实际数据也大致符合这个假设。另外数学上对正态分布的处理也最为成熟，基于该分布进行建模可行性是最高的，因此传统的量化金融模型大多会使用这一假设。但在实际生活中，市场并不总是服从正态分布的，这会导致传统量化模型在特定条件下表现很差。由于金融市场离钱最近，即使很小的模型偏差也会导致巨大而直接的金钱损失，这就是所谓的“黑天鹅”事件。所以现代量化金融的研究重点是利用非正态的概率分布（non Gaussian distribution）搭建模型。

[40] 在股票市场刚成立时，大部分上市公司是国有企业。这些企业大多数有 3 种类型的股份：国有股、法人股以及普通股。其中只有普通股能在股票市场上交易，于是这 3 种类型的股份形成了“同股不同权，同股不同利”的局面，也就是所谓的“股权分置”现象。

仅供非商业用途或交流学习使用

除了具体价格，股票市场每天的成交额（买卖股票的总金额）也是反映市场状态的一个很重要的指标。通常认为，在牛市里成交额会逐步放大，而在熊市里成交额会极度萎缩。针对 A 股市场，在使用成交额这个数据时需要特别注意这一点。1990 年至今（2017 年），中国经历了人类历史上罕见的经济高速发展期。伴随着经济的快速增长，通货膨胀是不可避免的，比如 1990 年末的货币供应量 M2 为 15 293.4 亿，而 2016 年末的 M2 为 1 550 100 亿元，增长了大约 100 倍。因此，在利用成交额搭建模型时，需要使用通货膨胀率对其做相应的折现处理[41]。在本节的例子中，我们使用的是 5 日成交额增长率以及 20 日成交额增长率，这就很巧妙地避开了上述的通货膨胀问题。

虽然 scikit-learn 并没有实现将要使用的 Gaussian HMM 模型[42]，但有比较成熟的开源方案：hmmlearn。它的安装方法很简单，只需运行“pip install -U --user hmmlearn”即可。

模型的代码实现很简单，如程序清单 9-6 所示。

（1）第三方库 hmmlearn 实现了 3 种用于非监督式学习的 HMM 模型，分别是 Gaussian mixture HMM、multinomial HMM 以及 Gaussian HMM。本节将使用 Gaussian HMM，它的具体实现为 GaussianHMM，如第 1 行所示。

（2）从原始数据中提取 4 个特征搭建模型，它们分别是 5 日的收益率（“r_5”）、20 日收益率（“r_20”）、5 日成交额增长率（“a_5”）以及 20 日成交额增长率（“a_20”），如第 7 行代码所示。

（3）GaussianHMM 里的参数“n_components”表示内在状态的个数，本例中这个值等于 3，如第 8 行代码所示。

参数“covariance_type”表示先验分布里协方差矩阵的类型（在状态已知的情况下，特征服从正态分布）。这个值等于“full”表示在不同市场状态下协方差矩阵是不同的，而且可以是任意半正定对称矩阵（也就是说并不要求协方差矩阵是对角矩阵），这和 9.3.5 节中讨论的二次判别分析是类似的。其他可能的取值为“spherical”（协方差矩阵为单位矩阵）、“diag”（协方差矩阵为对角矩阵）以及“tied”（不同状态下的协方差矩阵是一样的，这与 9.3.1 节中讨论的线性判别分析是一致的）。

参数“n_iter”则表示 EM 算法的最大循环次数。

（4）GaussianHMM 的模型 API 与 scikit-learn 非常相似，如第 9、10 行代码所示，使用 fit 方法训练模型；使用 predict 方法对数据做预测。

程序清单 9-6 Gaussian HMM

```
1 | from hmmlearn.hmm import GaussianHMM
```

[41] 折现就是利用折现率（可以理解为更广义的通货膨胀率），将现在的货币折算成等价的过去货币。举个简单的例子，假设当前时间距离过去的基准时间点为t年，需要折现的金额为a，折现率为β，则折现后的金额为$a\beta^t$。

[42] 事实上，较早版本的 scikit-learn（在 0.17 之前）是包含隐马尔可夫模型的，但后来因为项目目标的变化，这部分代码被移出了 scikit-learn，变成了本节将要使用的 hmmlearn。

仅供非商业用途或交流学习使用

```
def getHiddenStatus(data):
    """
    使用 Gaussian HMM 对数据进行建模，并得到预测值
    """
    cols = ["r_5", "r_20", "a_5", "a_20"]
    model = GaussianHMM(n_components=3, covariance_type="full", n_iter=1000)
    model.fit(data[cols])
    hiddenStatus = model.predict(data[cols])
    return hiddenStatus
```

根据模型的分类结果，将不同的状态的上证指数表示在不同的坐标系里可以得到如图 9-27 所示的结果，其中标记 1 为所有的市场交易数据，而标记 2、3、4 则是分类结果。图 9-27 表示模型的分类结果基本符合预期，比如在标记 2 中，股市上涨和下跌的比例大致相同，而且都在一个比较窄的区间内变化，因此对应的是震荡这个市场状态。类似地，可以推断出标记 3 表示的是熊市，而标记 4 表示的是牛市[43]。

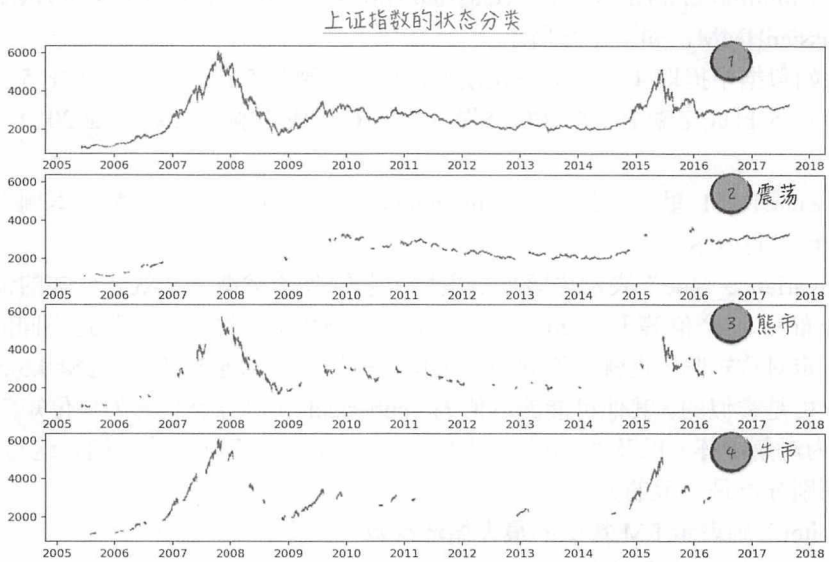


图 9-27[44]

基于上面的模型结果，很容易就能得到较为有效的量化交易策略，比如执行这样的策略就能获得不错的收益：若当天的隐藏状态是牛市（根据模型结果），则买入或继续持有股票，否则卖出股票或保持空仓。

[43] 虽然 Gaussian HMM 能将数据分类，但与监督式学习有所不同的是，分类结果并没有直接的物理意义。也就是说，模型只能将数据分类，但无法告诉我们每个类别的具体含义。因此需要人为地根据各个类别里的数据特点，给类别赋予含义。

[44] 完整的实现请参考随书配套的代码/ch09-generative_models/gaussian_hmm/stock_analysis.py。

仅供非商业用途或交流学习使用

如果实际生产中真的使用这个模型来做交易，你会沮丧地发现实际效果远不如历史数据的测算结果。这是因为隐马尔可夫模型的预测公式如下所示（程序清单 9-6 中第 10 行代码中的 predict 方法就是这么计算的）：

$$\hat{\boldsymbol{Y}} = \operatorname{argmax}_{\boldsymbol{Y}} P(y_0, y_1, \ldots, y_T \mid \boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_T) \tag{9-41}$$

也就是说，在测算历史数据时，预测变量y_1时，就已经知道了后来的实际交易数据$\boldsymbol{X}_2, \ldots, \boldsymbol{X}_T$。这一点在实际生活中是不可能实现的，因为没有办法预知未来。因此，为了使模型的测算结果更加符合现实，需要循环地调用公式（9-41）来预测y_T，而不是一次性地得到全部数据的预测结果。

9.5 本章小结

本章所讨论的是生成式模型。与之前章节讨论的判别式模型不同，生成式模型并不是从自变量出发对被预测量搭建模型。它的建模理念是通过模型理解数据是如何产生的，并以此为基础，借助贝叶斯框架对未知数据做预测。由此可见，生成式模型的建模过程更为复杂，这导致模型在数学上的处理也更为繁琐，因此本章出现了不少的数学公式。这些数学式子虽然“长得很吓人”，但体现的含义往往很直接，也是理解模型的关键，值得细细品味。

具体地，本章从简单到复杂，讨论了 3 类模型：朴素贝叶斯、判别分析和隐马尔可夫模型。

朴素贝叶斯的模型假设最强，它假设特征是条件独立的。虽然模型假设很牵强，但朴素贝叶斯在文本分类上的效果却不错，而且因为模型够简单，朴素贝叶斯常作为原子模型与其他模型进行联结组成更加复杂的模型，比如 multinomial HMM。

判别分析放松了特征条件独立的假设，允许自变量间相互关联。但这类模型假设特征在类别已知的条件下服从正态分布，因此它只能处理连续型变量。在实际中判别分析的常用方法有两种：一是对数据进行降维，二是与其他模型联结成复杂模型，比如 Gaussian HMM。

隐马尔可夫模型是一个复杂的图模型，这类模型在其他简单生成式模型的基础上加入马尔可夫链，因此它能处理另外两类模型处理不了的序列数据。不仅如此，隐马尔可夫模型还能同时应用于监督式学习和非监督式学习。在非监督式学习中，这类模型似乎能自动学习人类都无法观测到的被预测量，仿佛模型也有独立的智慧一样。这是一件非常神奇的事，也是人工智能吸引人的地方。

生成式模型是机器学习中非常热门的前沿领域，限于篇幅，本章只介绍了其中最基础也是最经典的内容，希望了解更多的读者可以参考 OpenAI 的官方网站。截至本章，本书的讨论重点是监督式学习，也就是说，模型要求训练数据里有标签变量（虽然本章的隐马尔可夫模型是个例外）。下一章将讨论一类全新的非监督式学习，这类模型能在没有标签变量的情况下，学习数据里隐含的相关关系。

仅供非商业用途或交流学习使用

第10章

非监督式学习：聚类与降维

其实地上本没有路，走的人多了，也便成了路。

——鲁迅

10.1 K-means

10.2 其他聚类模型

10.3 Pipeline

10.4 主成分分析

10.5 奇异值分解

10.6 本章小结

仅供非商业用途或交流学习使用

之前章节讨论的重点是监督式学习，这类模型的特点是所用的训练数据里有标签变量，通俗地讲就是数据里既有自变量$\boldsymbol{X}$，也有因变量y。这种情况下，搭建模型的思路是非常直观的，通过模型学习历史数据里变量$\boldsymbol{X}$与变量y之间的相关关系，并以此为基础，对未知数据做预测。模型搭建好之后，评估模型效果的思路也很直接，只需计算真实值与模型预测值之间的差异即可。

但在现实生活中的很多场景里，数据里是没有标签变量的，只有自变量$\boldsymbol{X}$。一方面是因为在某些场景中，被预测值y只是一些模糊的概念，没有办法被量化。比如我们希望通过一个人的行为预测他的性格，而性格本身就是没有清晰定义的，当然也就不可能收集到一个人的性格数据；另一方面是因为有时候收集被预测值y的难度很大，比如在零售行业，顾客对某件商品的偏好程度和能承受的最高价格等数据是很难被收集到的。

在这些场景下，之前讨论的监督式学习模型都不再适用了，因为没有标签变量，监督式模型也就无从学习了。虽然没有了变量y，但数据的自变量$\boldsymbol{X}$本身也包含很多有用的信息，很值得用模型去分析和学习。本章将重点讨论的非监督式学习就是为这个目的而设计的，这类模型能在没有“明确答案”的情况下，学习数据中的相关关系，并由此猜测“可能的答案”。

常用的非监督式学习可以分为两类，一类是聚类模型，另一类是降维模型，如图 10-1 所示。聚类模型会根据数据之间的相似度，将相似数据划分为一类；而降维模型是将高维空间里的数据映射到低维空间，这样可使我们更好地专注于数据的主要特征。

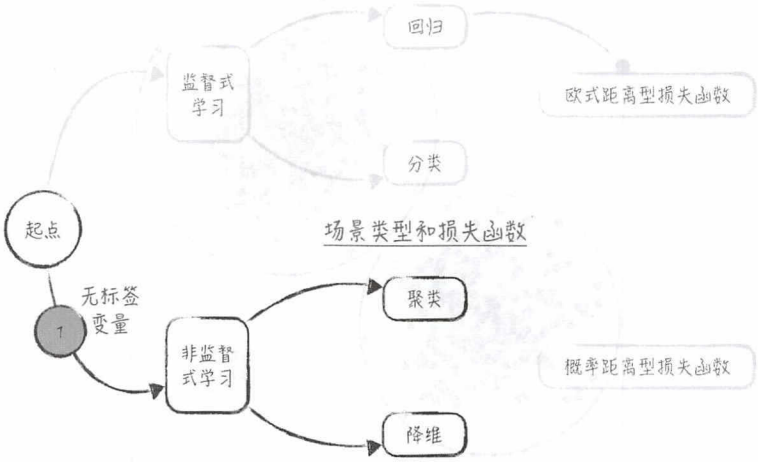


图 10-1

仅供非商业用途或交流学习使用

10.1 K-means

k-means 也称 k-平均算法，是一个十分简单的聚类算法。这个算法的思路非常简单明了，它根据数据之间的欧式距离将数据分为k类。下面来讲述这个模型的细节。

10.1.1 模型原理

为了更形象地理解模型，我们先来看一个简单的例子。假设需要聚类的数据是二维的，将它表示在坐标系中可以得到如图 10-2 所示的图像。

现在需要将这些数据分为两类，由于没有标注变量，因此无法确切地知道数据的类别，也无法用之前章节讨论的分类模型来学习类别与数据之间的关联关系。从直觉上来讲，离得越近的数据应该越相似，它们理应被划分为同一类别。将这个直觉翻译成数学语言就是，数据之间的相似度与它们之间的欧式距离成反比。这其实也是 k-means 模型的假设。

有了相似度的假设之后，将数据分为两类的算法就很明确了，尽可能将离得近的数据划分为一类。从图像直观上来看，就是用半径尽可能小的两个圆圈分别圈住数据，如图 10-2 中的黑色圆圈所示。

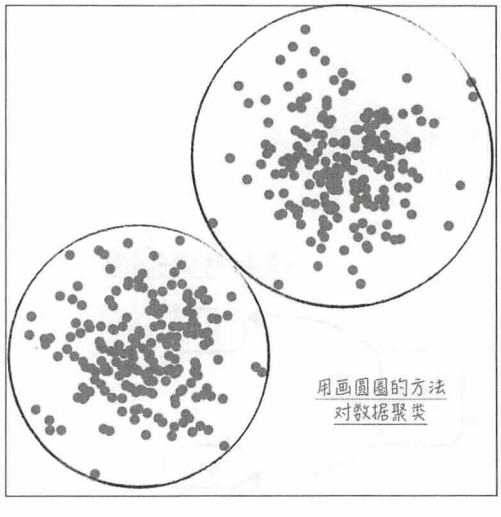


图 10-2

将上面的例子直觉抽象为数学定义，使之能应用到更一般的情况。为此，不妨假设需要将数据$\{\boldsymbol{X}_i\}$聚为k类（k值为给定的聚类个数，它的取值方法请参考 10.1.3 节），经过聚类后，每个

数据所属的类别记为$\{t_i\}$，而这k个聚类的中心被记为$\{\boldsymbol{\mu}_i\}$。由于数据间的欧式距离与数据间的相似度成反比，所以希望数据到它所属类别中心的距离越小越好。于是定义如下的损失函数：

$$L = \sum_{j=1}^{k}\sum_{i=1}^{n}(\boldsymbol{X}_i - \boldsymbol{\mu}_j)^2 1_{\{t_i = j\}} \tag{10-1}$$

k-means 模型的目的就是找到最佳的$\{t_i\}$，使得公式（10-1）所示的损失函数达到最小值（有了$\{t_i\}$之后，聚类中心$\{\boldsymbol{\mu}_i\}$可直接计算得到）。由此可以看到，$\{t_i\}$既是聚类的最终结果，也是需要估算的模型参数，它的估算公式如下：

$$\hat{t}_i = \operatorname{argmin}_{t_i} L \tag{10-2}$$

10.1.2 收敛过程

在 k-means 的损失函数中有两类未知参数，一类是每个数据所属的类别$\{t_i\}$，另一类是每个类别的中心$\{\boldsymbol{\mu}_i\}$。这两类未知参数是相互依存的：如果知道每个数据所属类别，那么类别中心就等于这个类别中所有数据的平均值；反过来，如果知道了类别中心，那么一个数据属于哪个类别取决于它离哪个中心点最近。这启发我们使用 9.4.6 节中讨论的最大期望算法（Expectation-Maximization Algorithm， EM）来估计 k-means 的模型参数，具体的步骤如下。

- 首先随机生成k个聚类中心点。
- 根据已有的聚类中心点，将数据分为k类。分类的原则是数据离哪个聚类中心最近，它就被分为哪一类。这一步就是 EM 算法中的 E step。
- 根据分类结果，重新计算每个聚类的聚类中心。这一步是 EM 算法中的 M step。
- 不断重复上述的 E step 和 M step，直至聚类中心收敛（聚类中心不再变动）。

以二维空间中的例子来展示这个收敛过程，可以得到如图 10-3 所示的图像。在图中的标记 1、2、3 中聚类结果和聚类中心并不匹配，也就是说，根据图中的聚类中心会得到不同的聚类结果。这说明算法还没有收敛，需要进一步迭代。而在标记 4 中，算法达到了收敛状态，因为聚类中心和聚类结果是相互吻合的。

由于 k-means 算法中的第一步是随机产生初始的聚类中心，因此对同一批数据，多次使用 k-means 算法进行聚类，可能会得到不同的聚类结果，即 k-means 的预测结果并不稳定。从技术上来讲，导致这种结果的原因通常有两个，一是因为算法收敛到了局部极值点而非全局最值点；二是因为算法在到达收敛点之前就停止迭代了。

当然这并不是 k-means 所独有的，事实上几乎所有的模型都有这样的问题：使用同样的数据多次训练模型，每次得到模型结果有可能并不一样。这是由于机器学习模型的参数都是由最优化算法估算的，而这些算法的计算过程都或多或少地有一些随机性。比如第 6 章中讨论的随机梯度下降法（stochastic gradient descent），这个算法的初始点就是随机生成的。而且由于最优化算法通常无法从理论上保证一定求得函数的最小值，因此算法结果通常并不稳定，不能保证每次运算的结果一定是一样的。

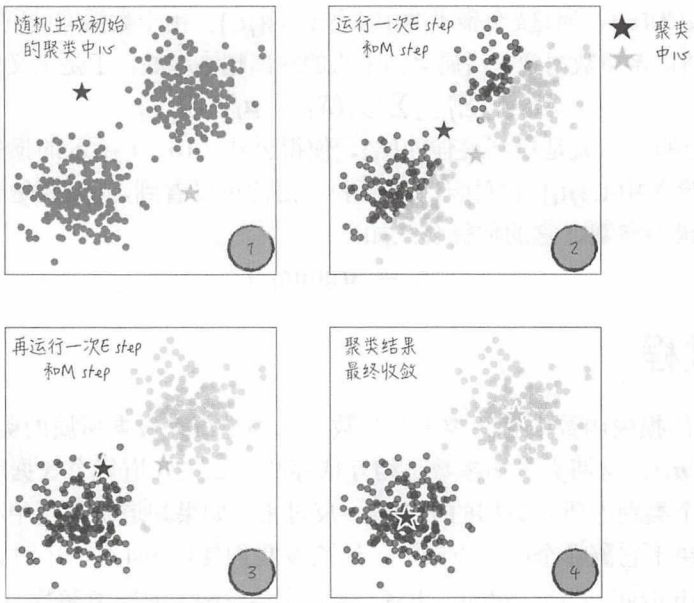


图 10-3[1]

为了解决k-means模型结果不稳定的问题，通常会反复多次地使用同一批数据训练模型，并从中选择效果最好的模型参数（其他模型解决模型结果不稳定的方法也是类似的）。这个过程的代码实现十分简单，如程序清单 10-1 所示。

（1）在第三方库 scikit-learn 中，非监督式学习的调用方法与监督式学习的调用方法是一样的，使用 fit 函数传入训练数据。唯一不同的是，对于非监督式学习，只需传入变量$\boldsymbol{X}$（因为没有变量y），如第 8 行代码所示。

（2）对于 KMeans 类，参数“n_clusters”表示聚类个数；参数“n_init”表示 k-means 算法的重复次数；参数“max_iter”表示最优化算法（EM 算法）的迭代次数；参数 algorithm 表示最优化算法的种类，该参数等于“full”时，表示将使用 EM 算法训练模型。

程序清单 10-1 k-means

```
1 | from sklearn.cluster import KMeans
2 |
3 | def trainModel(data):
4 |     """
5 |     使用 KMeans 模型对数据聚类
6 |     """
7 |     model = KMeans(n_clusters=2, max_iter=100, n_init=10, algorithm="full")
8 |     model.fit(data)
```

[1] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/kmeans.py。

```
    return model
```

10.1.3 如何选择聚类个数

本节将讨论聚类算法中非常重要的一个环节，如何评估聚类结果以及如何根据评估结果选择“较为正确”的聚类个数。

对于非监督式学习，训练模型的数据是没有标注变量的。那么除了极少数的情况，我们无从知道数据应该被分为几类。而且根据上面的讨论，对于 k-means 算法，给定一个聚类个数k，算法总能将数据分为k类。如图 10-4 所示，对于同样的数据，可以将它分别聚为 2 至 5 类。那么应该如何选择聚类个数k呢？在回答这个问题之前，先来看看应该如何评估一个聚类结果。这里讨论的评估方法有两个前提，一是数据满足 k-means 的所有模型假设，二是无法知道数据真实的类别。当这两个前提任意一个不满足时，都可能会出现评估指标很好，但实际聚类效果很差的情况，具体细节请参考 10.2 节。

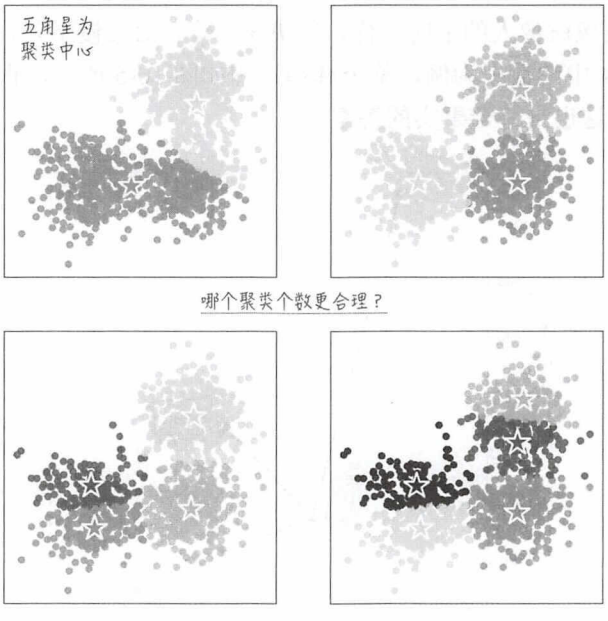


图 10-4[2]

上面的 10.1.1 节讨论了 k-means 算法的损失函数L，它等于每个点到相应聚类中心$\boldsymbol{\mu}_{t_i}$的距离平方和。

$$L = \sum_{j=1}^{k}\sum_{i=1}^{n}(\boldsymbol{X}_i - \boldsymbol{\mu}_j)^2 1_{\{t_i = j\}} \tag{10-3}$$

[2] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/kmeans_choose_k.py。

与第 4 章中的线性回归模型类似，这个值也是评价聚类结果的技术指标。事实上，可以从回归预测的角度来理解聚类模型。对于训练数据，聚类模型将它们划分为k个类别，同一个类别里所有数据的预测结果都是一样的，也就是聚类的中心。那么评价这个预测效果的技术指标自然是真实值到预测值的误差平方和，也就是L。

但与线性回归模型不同，对于 k-means，上面定义的评估指标是与具体的聚类个数相关的：很容易证明，聚类个数k越大，L也就越小。从某种意思上来讲，k值表示聚类模型的复杂度，跟其他机器学习模型一样，模型越复杂，对训练数据的预测效果也就越好，但也越容易引发过拟合（overfitting）问题（具体细节请参考 4.3.1 节）。因此需要在聚类个数k和预测误差平方和L之间做出某种妥协。

在实践中，通常采用所谓的 elbow method 来达到这个妥协的目的。这个方法并没有一个正式的中文翻译，但根据它所代表的算法，不妨叫它“手肘法”。这个方法假设当聚类个数小于真实的类别个数时，聚类结果的误差平方和会下降得很快，但当聚类个数超过真实值时，误差平方和虽然会继续下降，但下降速度会明显减缓。因此对于聚类算法，误差平方和随聚类个数变化的图像就像人的手肘一样，先快速下降，后缓慢下降，而转折点就是最佳的聚类个数。以图 10-4 中的数据为例，整个选择过程如图 10-5 所示，根据这个图的形状，最佳的聚类个数是 3，这也是符合事实的答案。

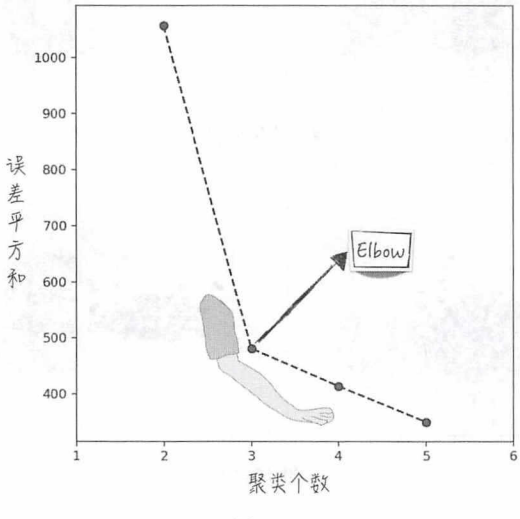


图 10-5

上面讨论的 elbow method 虽然十分直观，但很难用数学公式去量化它，因此这种方法使用起来的随意性较大。为了解决这个问题，学术界提出了一种数学上更严谨的方法来决定聚类的个数，这个方法被称为 silhouette analysis（不妨将其翻译成“轮廓分析”）。silhouette

analysis 的思路是计算聚类中心能在多大程度上代表这个类别里的所有数据[3]。限于篇幅，算法的细节在此就不做讨论了，有兴趣的读者请参考维基百科上的介绍。

10.1.4 应用示例

根据上面的讨论，k-means 模型能将相似的数据聚为一类，因此聚类的中心在某种程度上能代表这个类别里的数据。这个特性在实际生产中衍生出两类常见的应用：异常检测（anomaly detection）和图片压缩。

1．异常检测

顾名思义，异常检测的目的是要将那些明显有别于正常数据的异常值找出来，它在反欺诈、网络安全等领域有着相当广泛的应用。事实上，如果我们已经实现知道数据里的异常情况是什么样的，也就是说有比较明确的异常数据标签变量，那么可以使用之前章节讨论的监督式模型对数据进行建模。但不幸的是，大多数应用场景并没有这样的标签变量，而且很有可能我们对异常数据一点概念都没有，不知道它们会有什么样的特征。在这种情况下，k-means 以及其他聚类模型就能发挥出它们的独特优势了。一方面，聚类模型是非监督式学习，在没有标签变量的情况下也能工作；另一方面，由于聚类中心反映了相应类别的主要特点，因此，两个聚类中心之间的距离就可以衡量这两个类别之间的整体差异。于是就可以把原问题简化为从聚类中心中找出异常值，而后者通常是比较容易解决的，因为聚类中心的数量相对较少。

2．图片压缩

在图像处理中，常常用 k-means 聚类中心来代替原始像素点，这样既保留了图片中的主要特征，也有效地对其进行了压缩。这种方法在学术上被称为向量量化（Vector Quantization，VQ）。为了突出算法的重点，下面以黑白图片为例，说明向量量化的具体步骤，如图 10-6 所示。

（1）图像是由一个一个像素点组成的，图 10-6 中“图片”里的一个小方格就表示一个像素点，而这些像素点的取值是 0～255 的整数[4]。

---

[3] 在 silhouette analysis 中，假设对于一个聚类结果，a表示一个类别内部，所有数据到聚类中心的距离平方和，而b表示数据到最近其他聚类中心的距离平方和，则定义如下的 silhouette coefficient（用变量s表示）。

$$s = \frac{b - a}{\max(a, b)}$$

s的取值范围是$[-1, 1]$，当$s = 1$时，表示聚类的结果达到了最完美的状态；而$s = -1$表示聚类结果是最糟糕的状态。而且容易证明，s的取值与聚类个数没有直接的单调相关关系。因此通常来说，s的值越大表示聚类结果越好。这也是 silhouette analysis 被用来选择聚类个数的原因。

[4] 对于黑白图片，像素点的值是一维的，取值表示该点的黑白度。对于彩色照片，像素点的值是三维的，分别表示红、绿、蓝 3 种颜色（RGB）的程度，但每一维的数值仍然是 0～255 的整数。

（2）首先通过4 × 4相互不重叠的小格子将图像转换为一系列 16 维向量。

（3）使用 k-means 模型对这些向量做聚类运算，找出相应的聚类中心，这一步被称为编码（encode）。

（4）在聚类结果的基础上，用相应的聚类中心代替原始像素点的取值，这一步被称为解码（decode）。

（5）使用解码后的数据，重新构建图片。

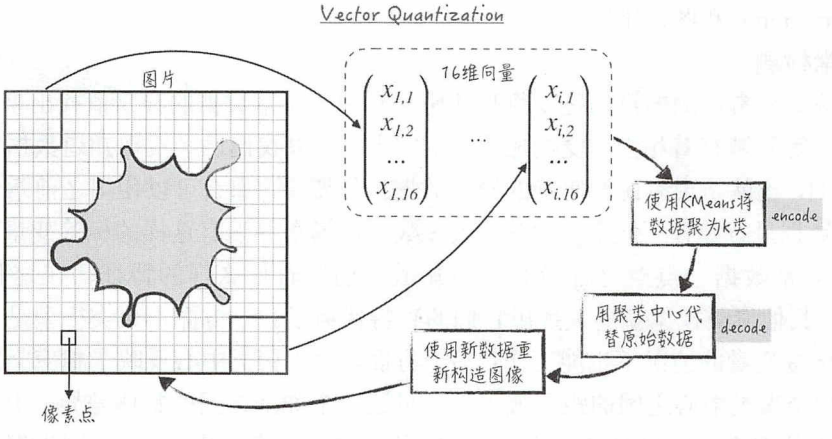


图 10-6[5]

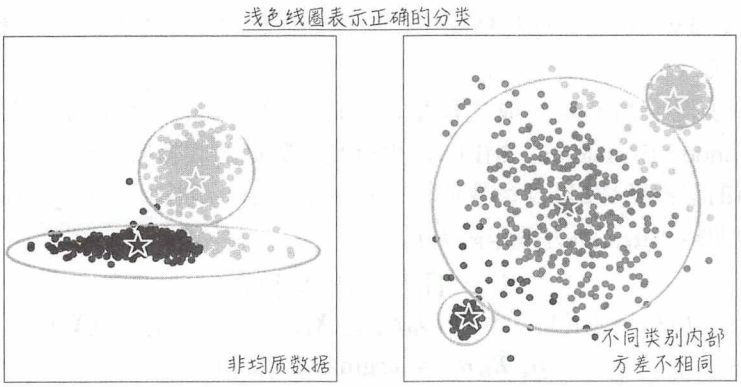
10.2 其他聚类模型

k-means 是非常简单的聚类模型，但它有两个比较明显的缺陷，或者更准确地说，k-means 有两个很强的模型假设，而这两个假设在很多应用场景中无法被满足。

k-means 模型用欧式距离来衡量数据间的相似程度，因此它要求数据在各个维度上是均质的。以二维空间为例，这表示相似数据大概能被圆圈圈住，而在图 10-7a 中，数据是非均质的，因此 k-means 的聚类效果并不好。

另外，在 k-means 的损失函数中，不同类别的权重是一样的，即模型假设不同类别的内部方差是大致相等的。以二维空间为例，这表示用来圈住不同类别的圆圈是差不多大小的，而在图 10-7b 中，3 个类别的内部方差是不一样的，因此使用 k-means 的聚类效果也不理想。

[5] 图片参考自 Trevor Hastie 等人编写的 *The Elements of Statistical Learning*。



（a）　　（b）

图 10-7[6]

上面这两种数据的聚类问题，可以用其他聚类模型很好地解决，下面将讨论这些模型的细节。

10.2.1 混合高斯之模型原理

混合高斯（Gaussian Mixture Model，GMM）是一种生成式模型（generative model）。在 9.3.5 节中，我们讨论了二次判别分析（Quadratic Discriminant Analysis，QDA），这两个模型其实是同一个模型。也就是说，这两个模型的假设和推导过程都是一模一样的，只是混合高斯是非监督式学习，而二次判别分析是监督式学习。正如 9.4.6 节中讨论的，这其实是生成式模型的一大特性，或者说一大优点：同样的模型假设既可以处理有标签变量的数据，也可以处理无标签变量的数据[7]。为了更好地理解这一点，下面先来看看混合高斯的模型假设。

假设现在将使用混合高斯将数据聚为两类，则模型假设的先验概率（prior probability）如公式（10-4）所示，其中，$\mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$表示期望为$\boldsymbol{\mu}_i$，协方差为$\boldsymbol{\Sigma}_i$的正态分布。

$$\begin{aligned} \boldsymbol{X} \mid y &= 0 \sim \mathcal{N}(\boldsymbol{\mu}_0, \boldsymbol{\Sigma}_0) \\ \boldsymbol{X} \mid y &= 1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) \\ P(y = 0) &= \theta_0, P(y = 1) = \theta_1 \end{aligned} \tag{10-4}$$

如果训练模型的数据里既有变量$\boldsymbol{X}$，又有变量y，则可以根据数据直接估算公式（10-4）中的模型参数$\{\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i, \theta_i\}$，这其实就是二次判别分析模型。但混合高斯面对的训练数据里只

[6] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/kmeans_limitations.py。

[7] 从技术的角度来讲，生成式模型能处理无标签数据是因为有最大期望算法。若抛开这些技术细节，从宏观的角度来讲，生成式模型的建模理念是首先“理解”数据是如何产生的，然后在这个基础上“再次生成”数据并预测结果。而标签变量的缺失，并不影响模型“理解”数据（虽然这确实增加了模型“理解”数据的难度）。因此生成式模型能同时适用于监督式学习和非监督式学习这两个应用场景。

有变量$\boldsymbol{X}$，没有变量y，那么在这种情况下，应该如何估算模型参数呢？答案就是上面讨论过的最大期望算法。

具体地，由于混合高斯是生成式模型，因此模型参数的估算原则是最大似然估计法（Maximum Likelihood Estimation，MLE），也就是使数据的联合概率达到最大值。根据模型假设，混合高斯的似然函数L以及参数估算公式如下所示，其中函数$f_{\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i}$是正态分布的概率密度函数，它的期望为$\boldsymbol{\mu}_i$，协方差矩阵为$\boldsymbol{\Sigma}_i$。

$$L = \prod_i P(y_i)P(\boldsymbol{X}_i \mid y_i)$$

$$\ln L = \sum_i (1 - y_i)\ln\theta_0 f_{\boldsymbol{\mu}_0,\boldsymbol{\Sigma}_0}(\boldsymbol{X}_i) + y_i \ln\theta_1 f_{\boldsymbol{\mu}_1,\boldsymbol{\Sigma}_1}(\boldsymbol{X}_i)$$

$$\hat{\boldsymbol{\mu}}_i, \hat{\boldsymbol{\Sigma}}_i, \hat{\theta}_i = \operatorname{argmax}_{\boldsymbol{\mu}_i,\boldsymbol{\Sigma}_i,\theta_i} \ln L \tag{10-5}$$

反过来，如果知道了模型参数，聚类的结果$\hat{y}_i$可以由最大后验概率（posterior probability）得到（具体细节请参考 9.1.4 节），如公式（10-6）所示。

$$\hat{y}_i = \operatorname{argmax}_{y_i} P(\boldsymbol{X}_i \mid y_i)P(y_i) \tag{10-6}$$

基于这些公式，最大期望算法的具体步骤（针对混合高斯模型）如图 10-8 所示。

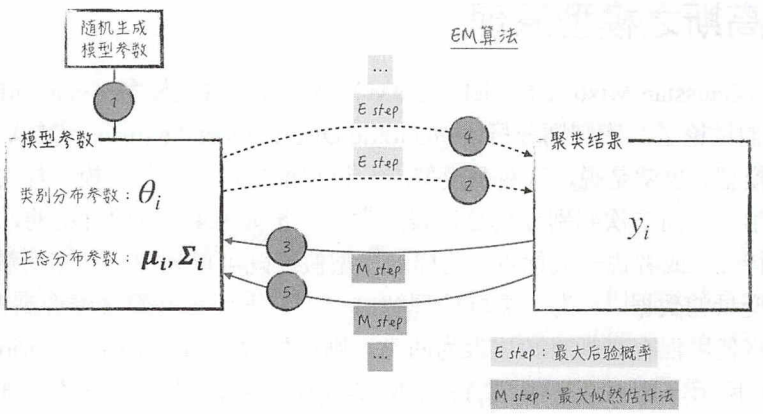


图 10-8

10.2.2 混合高斯之模型实现

使用混合高斯对图 10-7 中的数据进行聚类，可以得到如图 10-9 所示的结果，可以看到混合高斯的聚类结果更加合理。

从直观上来讲，二维空间里的混合高斯模型就是用椭圆形状（圆是一种特殊的椭圆）的线框去圈住数据，而 k-means 是用圆圈去圈住数据，因此，k-means 其实是一种特殊的混合高斯模型[8]。

[8] 从数学上很容易证明，如果混合高斯模型里的协方差矩阵都为单位矩阵，则它等同于 k-means 模型。

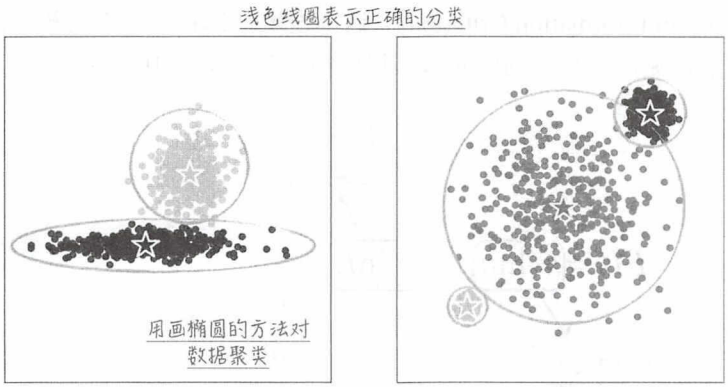


图 10-9[9]

混合高斯的代码实现很简单，如程序清单 10-2 所示。

（1）在 scikit-learn 中，混合高斯的实现为 GaussianMixture，如第 1 行代码所示。它的调用方式和 k-means 类似，通过 fit 函数传入训练数据，如第 8 行代码所示。

（2）对于混合高斯模型，比较重要的参数有两个，“n_components”表示聚类的个数；“covariance_type”表示正态分布里协方差矩阵的类型。它的取值有 4 个：“full”表示不同类别的协方差矩阵是不同的，而且可以是任意的半正定对称矩阵；“tied”表示不同类别的协方差矩阵是一样的；“diag”表示协方差矩阵是对角矩阵，但不同类别的协方差矩阵是不同的；“spherical”表示协方差矩阵是单位矩阵乘以一个常数。

程序清单 10-2 GMM

```
1  from sklearn.mixture import GaussianMixture
2
3  def trainModel(data, clusterNum):
4      """
5      使用混合高斯对数据进行聚类
6      """
7      model = GaussianMixture(n_components=clusterNum, covariance_type="full")
8      model.fit(data)
9      return model
```

与 k-means 类似，混合高斯也面临着如何选择聚类个数的问题。为了解决这个问题，与 k-means 类似，首先需要定义评估模型的技术指标。由于混合高斯是生成式模型，它估计模型参数的原则是使似然函数达到最大值。从概率上来讲，似然函数对应着数据出现的联合概率，因此学术界在似然函数的基础上，定义了评估混合高斯模型的技术指标：BIC[10]（贝叶

[9] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/gmm.py。

[10] 除了 BIC 外，还有一种常用的技术指标叫 AIC（Akaike Information Criterion）。它的定义与 BIC 非常类似：$AIC = 2k - 2\ln L$，公式中字母的具体含义请参考图 10-10。

书籍是人类进步的阶梯

斯信息准则，Bayesian Information Criterion）。这个技术指标将综合考虑模型的似然函数和模型复杂程度（为了避免过度拟合的问题），具体的定义如图 10-10 所示。

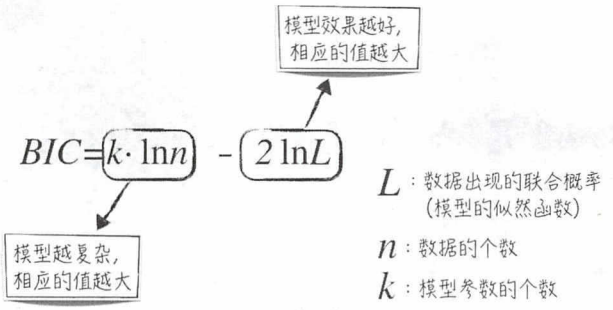


图 10-10

由于模型参数的个数可以衡量模型的复杂程度，因此可以得到，对于同样复杂的几个模型，模型效果越好，则相应的 BIC 值越小。类似地，对于同样预测效果的几个模型，模型复杂程度越低，则相应的 BIC 值越小。由此可以总结出 BIC 使用的原则是选择 BIC 值最小的模型，具体到混合高斯模型，就是选择 BIC 值最小的聚类个数。下面以图 10-7b 中的数据为例，若通过 BIC 指标来选择聚类个数，可以得到如图 10-11 所示的结果：聚类个数等于 3 且协方差矩阵类型等于“spherical”时，BIC 指标达到最小值。

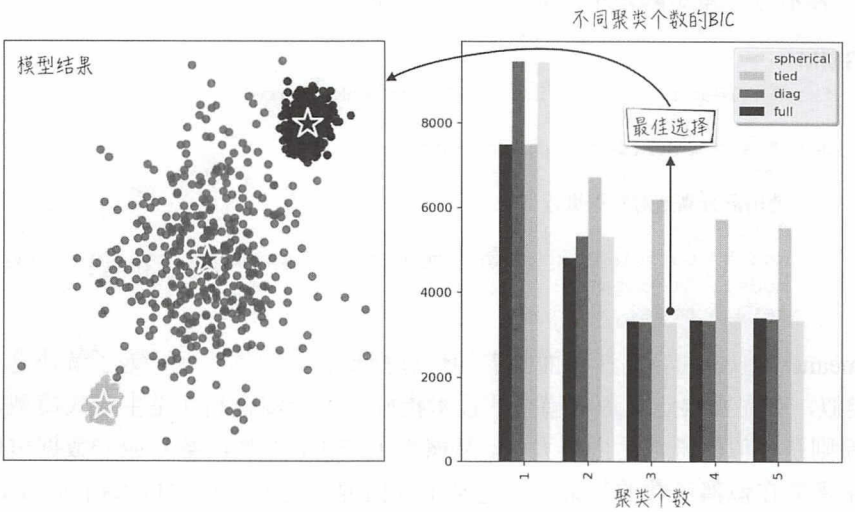


图 10-11[11]

[11] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/gmm_choose_k.py。

书籍是人类进步的阶梯

10.2.3 谱聚类之聚类结果

以二维空间为例，混合高斯的分类方法可以直观地理解为用不同形状的椭圆去圈住一个聚类的数据。虽然这个模型解决了很多 k-means 无法解决的问题，但它有一个比较明显的缺陷，无法对弯曲的数据进行有效的聚类，如图 10-12 所示。

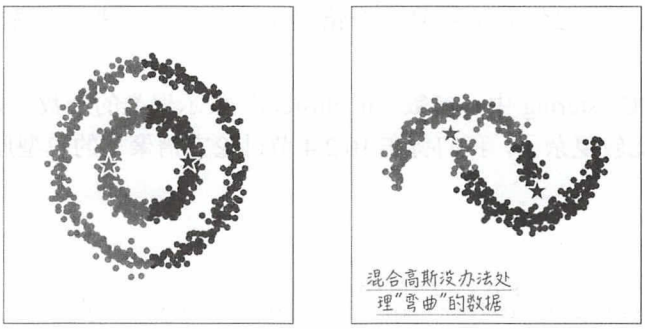


图 10-12

为了能对上面的数据进行有效的聚类，我们引入谱聚类（spectral clustering），它的聚类结果如图 10-13 所示，可以看到谱聚类能很好地捕捉数据中的弯曲特性。

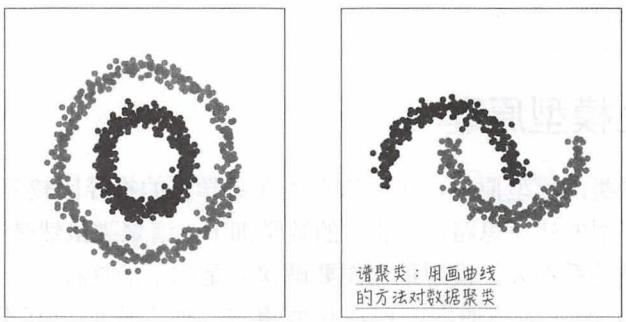


图 10-13[12]

与其他聚类模型类似，scikit-learn 也实现了谱聚类。但遗憾的是，scikit-learn 的实现中有一个很致命的数学错误[13]（截止到本书编写时的 0.18 版本），导致得到的结果完全背离模

[12] 图 10-12 和图 10-13 的实现请参考随书配套的代码/ch10-unsupervised/clustering/spectral_clustering/gmm_vs_spectral.py。

[13] 这个错误的细节并不是这里讨论的重点，但对细节有兴趣的读者请参考 GitHub 网站上 scikit-learn 的 issue-8129。

书籍是人类进步的阶梯

型设计的初衷。为了展示正确的谱聚类结果，我们在 scikit-learn 的基础上做修正，实现正确的谱聚类模型。限于篇幅，修正的细节在此就不做讨论了（对工程实现感兴趣的读者请参考随书配套的代码/ch10-unsupervised/clustering/spectral_clustering）。我们实现的模型接口与 scikit-learn 中的完全一样，因此调用模型的方法是一致的，如程序清单 10-3 所示。

（1）我们在 spectral.py 中重新实现了谱聚类模型，因此引用模型的方式有所不同，如第 1 行代码所示。如果要使用 scikit-learn 中的实现，则引用的语句如第 2 行代码所示（scikit-learn 的开源社区已经着手修复这个错误了，希望当读者读到本章时，这个 bug 已经不存在了）。

（2）在 SpectralClustering 中，参数“n_clusters”表示聚类的个数，如第 8 行代码所示。其他参数的意义就比较复杂了，我们将在 10.2.4 节讨论完谱聚类的模型原理后，再回过来介绍这些参数。

程序清单 10-3 谱聚类

```
 1 | from spectral import SpectralClustering
 2 | # from sklearn.cluster import SpectralClustering
 3 |
 4 | def trainSpectralClustering(data, clusterNum):
 5 |     """
 6 |     训练谱聚类模型
 7 |     """
 8 |     model = SpectralClustering(n_clusters=clusterNum, affinity="rbf",
 9 |         gamma=100, assign_labels="kmeans")
10 |     model.fit(data)
11 |     return model
```

10.2.4 谱聚类之模型原理

本节将讨论谱聚类的模型原理，由于谱聚类在数学上的推导比较复杂，为了便于理解，本节的讨论重点是模型的建模思路，而非它的数学细节。谱聚类虽然是一种聚类模型，但它的设计初衷却与聚类关系不大，模型最初想要解决的是如下的这样一个问题。

对于如图 10-14 所示的连通图，现在想要将它“剪”成两个互不相交的连通子图，那么什么样的划分方法是最有效的呢？这个问题需要从两个方面考虑，一方面我们希望被剪掉的边越小越好；另一方面，我们还希望得到的两个连通子图是几乎差不多大小的。将这两方面的考虑翻译成数学语言，假设将原始连通图划分为 1 和 2 两个子图，则定义如下技术指标$Ncut(1,2)$来衡量划分的质量[14]。其中，$cut(1,2)$表示子图 1 和子图 2 之间

[14] 这个技术指标在学术上被称为 normalized cut。具体的细节请参考 Shi J, Malik J. Normalized cuts and image segmentation[C]// Computer Vision and Pattern Recognition, 1997. Proceedings. 1997 IEEE Computer Society Conference on. IEEE, 1997:731-737.

书籍是人类进步的阶梯

书籍是人类进步的阶梯

边的个数（或者边的权重之和，如果边有权重的情况），比如图 10-14 中$cut(1,2) = 1$；$assoc(i)$表示与子图i相关的边数（内部边数加上外围边数），比如图 10-14 中，$assoc(1) = 7, assoc(2) = 4$。

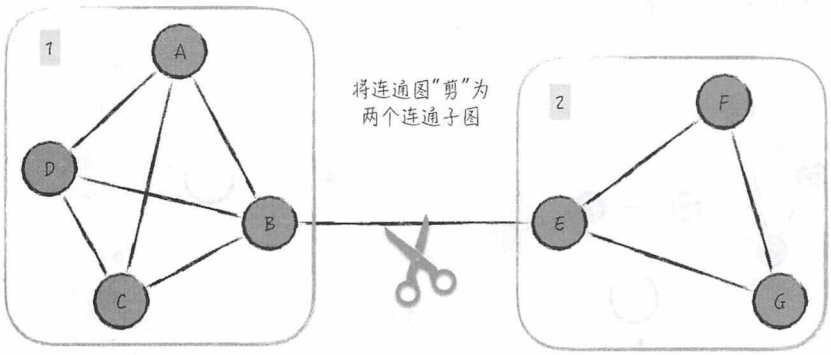


图 10-14

$$Ncut(1,2) = \frac{cut(1,2)}{assoc(1)} + \frac{cut(1,2)}{assoc(2)} \tag{10-7}$$

$Ncut(1,2)$的值越小，表示划分连通图时受到的阻力也就越小，最理想的划分方法将使$Ncut(1,2)$达到最小值。公式（10-7）虽然很简单，但求解它却相当困难：一方面，数学上并无有效的算法来求解这个问题；另一方面，穷举求解的运算量又太大，没有实用价值。但幸运的是，数学上可以证明，利用矩阵特征向量（eigenvector）可以近似地求解这个问题，具体步骤如下。

（1）首先将连通图转换为邻接矩阵（adjacency matrix）。对于有n个节点的图，相应的邻接矩阵为$n \times n$矩阵，记为$\boldsymbol{A} = (a_{i,j})$，元素$a_{i,j} = 1$表示节点$i$与节点$j$之间有边相连，否则$a_{i,j} = 0$（若连通图中的边有权重，则$a_{i,j}$等于节点$i$与节点$j$之间边的权重）。

（2）在邻接矩阵的基础上，定义 degree matrix，记为$\boldsymbol{D} = (d_{i,j})$和 Laplacian matrix，记为$\boldsymbol{L} = (l_{i,j})$。其中，$\boldsymbol{D}$为$n \times n$的对角矩阵，元素$d_{i,i}$表示第$i$个节点的连接边数，具体的公式为$d_{i,i} = \sum_{j=1}^{n} a_{i,j}$（对于有权重的连通图，$d_{i,i}$表示第$i$个节点的权重之和）；而 Laplacian matrix 的定义为$\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$。

（3）数学上可以证明，若用 Laplacian matrix 的特征向量（按特征值从小到大排列）来表示图中各节点[15]，则连接较为紧密的节点在转换后的欧式空间里也离得很近。这个方法在学术上被称为 spectral embedding（由于没有正式的中文翻译，不妨将其称为谱编码）。

[15] 假设连通图里有n个节点，则相应的 Laplacian matrix 为$n \times n$的矩阵。在 spectral embedding 中，我们通常会选取 Laplacian matrix 从小到大的前k个特征向量，这k个特征向量放在一起组成了$n \times k$的矩阵，这个矩阵的行向量就对应着连通图中的节点。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

（4）在 spectral embedding 的基础上，通常使用 k-means 方法等聚类模型对数据进行聚类，得到结果就为近似的最佳划分结果，也就是谱聚类的结果。以图 10-14 中的数据为例，具体的计算步骤如图 10-15 所示，为了直观地表示，我们选择其中最佳的两个特征向量（相应的特征值为最小的两个）来表示连通图中的各个节点。

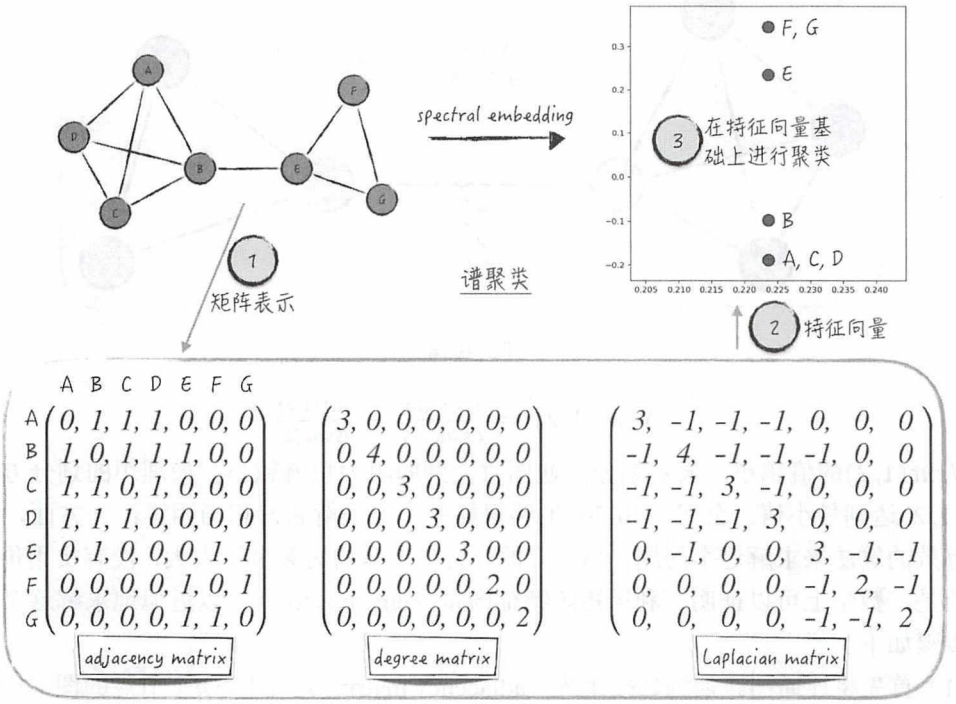


图 10-15[16]

总结上面讨论的模型步骤。在理论层面，谱聚类更像是一种模型的联结：首先通过 spectral embedding 算法，将连通图中的节点转换为向量，再通过经典的聚类模型来完成最后的聚类运算。其中第一个步骤，如何使用向量有效地表示节点（这个过程在学术上被称为向量化），是整个模型中最为关键的一步。可以用邻接矩阵中的行向量来表示图中的节点，这也是最直观的方法，但是这样的向量表示并没有很好地揭示出连通图的结构，因此无法直接对其做聚类运算；如果借助 Laplacian matrix 的特征向量表示节点，则得到的向量本身就反映了连通图中的结构信息，为后续的聚类运算提供了帮助。

稍稍扩展一下，在机器学习的很多应用场景中，将建模对象向量化是建模过程中最核心的一步。比如对于图像处理，用上面讨论的 spectral embedding 来表示图片中的像素点能较

[16] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/spectral_clustering/embedding_example.py。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

好地表示像素点的图像特征，具体的例子请参考 10.2.5 节。又比如针对自然语言处理，用 10.5.3 节将讨论的潜在语义分解来处理文本就能较好地用向量来表达文字的意思。在这些向量表示的基础上，可以很容易地复用经典的机器学习模型来解决问题，如图 10-16 所示。

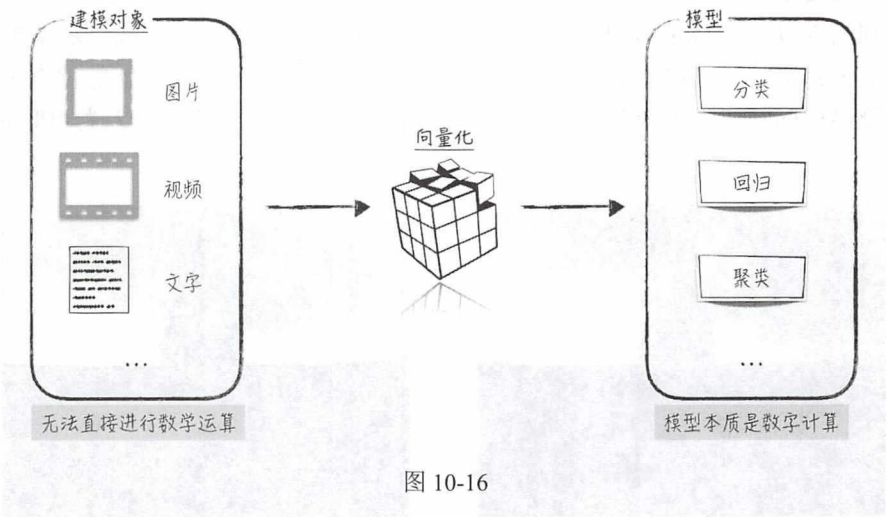


图 10-16

在实用层面，若需要使用谱聚类，则需要定义数据间的邻接矩阵。除了直接定义外，还可以使用其他方法来完成矩阵的定义。比如程序清单 10-3 中第 8 行代码中的“affinity='rbf'”就表示使用 RBF 核函数（细节请参考 8.2.5 节）来定义邻接矩阵，具体步骤如下。

（1）针对图 10-13 中的数据，图中的每一个点（相应的坐标记为$\boldsymbol{X_i}$）被表示为连通图中的节点i。

（2）如公式（10-8）所示，定义节点i与节点j之间边的权重，并由此定义邻接矩阵$\boldsymbol{A} = (a_{i,j})$：

$$a_{i,j} = \mathrm{e}^{(-gamma\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)} \tag{10-8}$$

定义好邻接矩阵之后，需要选择具体的聚类模型来完成最后一步的聚类运算，比如程序清单 10-3 中第 9 行代码中的“assign_labels='kmeans'”就表示将使用 k-means 模型。

10.2.5 谱聚类之图片分割

现在将讨论如何使用谱聚类进行图片分割（image segmentation），这也是谱聚类模型的设计初衷。所谓图片分割，就是根据图片中各种物体的形状，将图片划分若干个区域，每个区域只包含一个或多个相似的物体。从直观效果来讲，就好像机器能识别和理解图片的具体内容一样。为了避免陷入过多的细节，这里只讨论黑白图片的分割。

正如上面讨论的，使用谱聚类的第一步就是定义图片的邻接矩阵。由于图片是黑白的，我们用变量x_i表示第i个像素点的黑白程度。在连通图中，第i个像素点为第i个节点，而节点

书籍是人类进步的阶梯

i与节点j之间边的权重（邻接矩阵的元素）如公式（10-9）所示。

$$a_{i,j}=\begin{cases}e^{-\text{gamma}(x_i-x_j)^2} & ,\ i\text{与}j\text{相邻}\\ 0 & ,\ i\text{与}j\text{不相邻}\end{cases} \tag{10-9}$$

在这个邻接矩阵的基础上，使用谱聚类模型就可以完成图片分割。以图 10-17a 为例，若将其分割为 3 块，具体的效果如图 10-17b 所示，图中的虚线为分割线。从结果上来看，模型成功地将建筑物（巴黎荣军院）从图片中识别了出来。这就是人工智能神奇之处：通过一系列跟最终目标貌似不太相关的数学运算，让机器在某些方面获得和人类相似的智能。

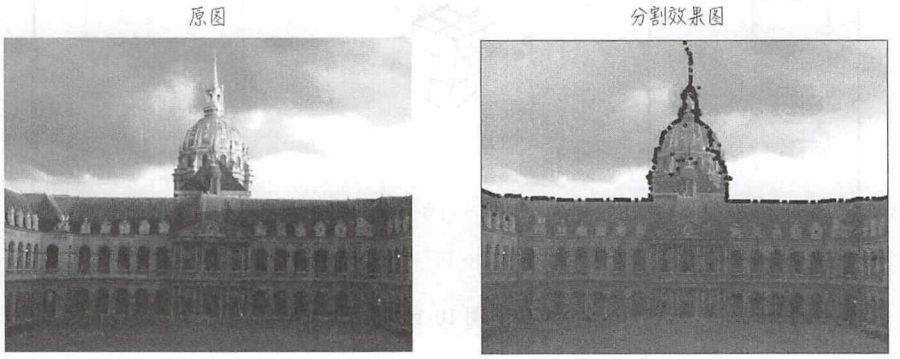


图 10-17[17]

10.3 Pipeline

在第三方库 scikit-learn 中，对于模型的工程实现，有一个很重要的实现——Pipeline（流水线）。我们在 9.2.5 节中已经初步接触过它，由于这个实现是 scikit-learn 最具特色也是最为出彩的一点，因此本节将详细讲解它的细节。

Pipeline 这个工程实现对应的理论基础是模型的联结主义（connectionism，细节请参考 7.6.5 节）。从理论上来看，本书讨论的大部分模型都可以视为原子模型，它们可以单独使用，但在实际生产中，它们更多的是作为整体模型的一部分，与其他模型组装起来使用。这样的描述可能有点空泛，现在来看一个具体的例子。正如上面讨论的，谱聚类其实是 spectral embedding 和 k-means 这两个模型的联结。

[17] 完整的实现请参考随书配套的代码/ch10-unsupervised/clustering/spectral_clustering/photo_segmentation.py。需要注意的是，这个脚本里使用了两个额外的第三方库。一个是读取图片的 pillow，它的安装命令为“pip install pillow”；另一个是加速模型训练的 pyamg，它的安装命令为“pip install pyamg”。

书籍是人类进步的阶梯

从工程实现的角度来讲，我们会首先调用 spectral embedding 对训练数据做特征提取，再调用 k-means 完成最后的模型运算[18]，而这正是 Pipeline 的设计理念。

- 一个 Pipeline 由n个模型按顺序组成，其中前$n-1$个模型被称为 Transformer，主要作用是对数据进行特征提取，最后一个模型被称为 Estimator，主要作用是在特征的基础上完成最后的模型预测。
- 从代码层面上来讲，前面的$n-1$个 Transformer 必须实现 fit 和 transform 这两个接口，最后一个 Estimator 则只需实现 fit 这个接口，它们之间的依赖关系如图 10-18 所示。

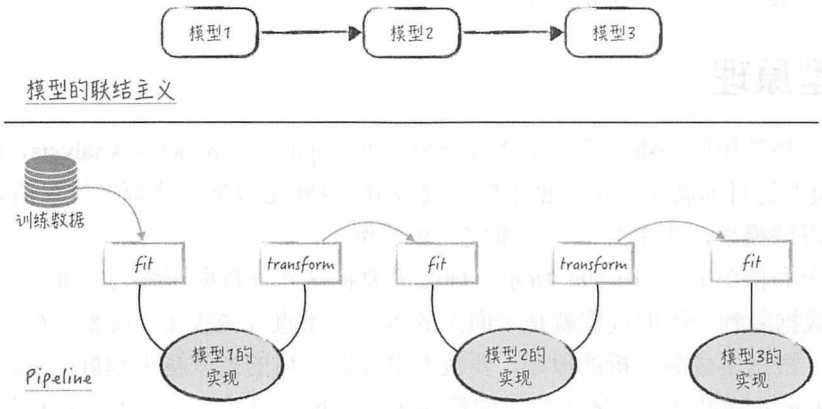


图 10-18

Pipeline 的代码实现十分简单和直观，具体细节请参考 9.2.5 节中的程序清单 9-3，在此不再赘述。

10.4 主成分分析

讨论完聚类算法后，现在开始讨论非监督式学习里另一类很重要的模型——降维（dimension reduction），也就是将高维空间里的数据映射到低维空间。比如假设在三维空间中，某点 A 的坐标为(x_a, y_a, z_a)，则这 3 个坐标在一起表示了点 A 完整的空间信息。如果只使用(x_a, y_a)这两个坐标，也可以部分表示点 A 的信息，事实上(x_a, y_a)表示的是点 A 在xy平面的投影。如上所述，从三维向量(x_a, y_a, z_a)到二维向量(x_a, y_a)的过程就是一种降维。

[18] 在代码层面，scikit-learn 对 spectral embedding 的实现为 sklearn.manifold.SpectralEmbedding。这个类并没有实现 transform 方法，因为从模型上来看，spectral embedding 只能处理训练数据，无法处理未知数据。因此虽然 spectral embedding 和 k-means 的配合方式完成符合 Pipeline 的设计，但并不能通过现有的 Pipeline 将两者组装起来。

书籍是人类进步的阶梯

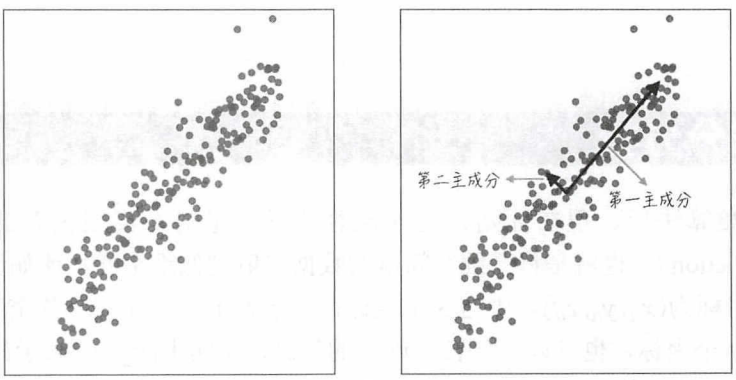
虽然对数据降维会不可避免地损失部分信息，但正如 9.3.3 节中讨论的，降维能帮助我们降低随机因素的干扰，从而更好地抓住数据的主要特征。另外基于降维的数学公式，可以倒推出数据里的隐含因素，在实际生产中，我们常基于这些挖掘出的隐含因素搭建语义理解模型和大型推荐系统，具体细节请参考 10.5 节。因此降维是机器学习里很重要也很常用的处理步骤。

虽然降维常被划分为非监督式学习，但其实它对数据里是否有标签变量并无要求，之前的 9.3.3 节就曾讨论如何使用线性判别分析对有标签变量的数据进行降维。但这种应用场景在实际中并不常见，常碰到的是在没有标签变量（或者无视标签变量）的情况下对数据进行降维，这也是本节和下一节讨论的重点。

10.4.1 模型原理

首先讨论最常用的降维模型，主成分分析（Principal Components Analysis，PCA）。顾名思义，这个模型的目的就是找出数据中的主要成分，也就是数据变化幅度排名前几位的维度。为了形象地理解模型，先来看一个二维空间里的例子。

数据的分布情况如图 10-19a 所示，现在需要将这部分数据压缩为一维。由于从直观上来讲，二维数据降到一维的过程就是平面上的点向某条直线做投影，投影后得到的点就是降维后的数据。根据主成分分析的原则，要数据求投影之后的方差越大越好。由此可以得到如图 10-19b 所示的降维结果，图中较长的箭头表示降维的直线，在学术上被称为第一主成分（first principal component），因为数据在这个方向上的变化幅度最大。类似地还可以定义第二主成分，也就是数据变化幅度排名第二位，且与第一主成分垂直的方向。



（a）　　（b）

图 10-19[19]

[19] 完整的实现请参考随书配套的代码/ch10-unsupervised/dimension_reduction/pca.py。

书籍是人类进步的阶梯

下面从数学层面讨论上面的降维结果是如何得到的，假设需要降维的数据有n个，数据维度位为m维，记第i个数据为$\boldsymbol{X}_i = (x_{i,1}, x_{i,2}, \cdots, x_{i,m})$。为了推导方便，不妨假设这$n$个数据的中心为原点[20]，也就是说$\sum_{j=1}^{n} x_{j,l} = 0;\ l = 1, \cdots, m$。

首先来看看将数据降到一维的情况，不妨假设降维直线的单位向量（也就是表示直线方向的向量）为$\boldsymbol{U} = (u_1, u_2, \cdots, u_m)$; $\|\boldsymbol{U}\|^2 = 1$，则第i个点在这条直线上的投影如公式（10-10）所示。其中，$\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}}$为投影的长度。

$$\widetilde{\boldsymbol{X}}_i = (\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}})\boldsymbol{U} \tag{10-10}$$

根据主成分分析的原则，我们希望降维后，数据的方差越大越好。注意到数据的中心在降维过程中不变，仍然为原点，因此降维后数据的方差如公式（10-11）所示。

$$V = \sum_{i=1}^{n} \left\|\widetilde{\boldsymbol{X}}_i\right\|^2 = \sum_{i=1}^{n} (\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}})^2 \tag{10-11}$$

注意到$\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}}$是一个实数，因此$(\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}})^{\mathrm{T}} = \boldsymbol{U}\boldsymbol{X}_i^{\mathrm{T}} = \boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}}$。根据这个式子，对公式（10-11）做进一步的变换得到公式（10-12）。

$$V = \sum_{i=1}^{n} \boldsymbol{U}\boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{U}\left(\sum_{i=1}^{n} \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_i\right)\boldsymbol{U}^{\mathrm{T}} \tag{10-12}$$

不妨将$\sum_{i=1}^{n} \boldsymbol{X}_i^{\mathrm{T}}\boldsymbol{X}_i$记为$\boldsymbol{C}$，它是一个$m \times m$的矩阵，对数学比较熟悉的读者会发现，$\boldsymbol{C}$其实是数据的协方差矩阵。那么主成分分析的降维原则是$\max_{\|\boldsymbol{U}\|=1} \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^{\mathrm{T}}$，由此可以得到最佳降维向量，也就是第一主成分的估算公式：

$$\widehat{\boldsymbol{U}} = \operatorname{argmax}_{\|\boldsymbol{U}\|=1} \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^{\mathrm{T}} \tag{10-13}$$

对于公式（10-13）所示的最优化问题，数学上可以证明，当$\boldsymbol{U}$为矩阵$\boldsymbol{C}$的特征向量（eigenvector）且相应的特征值（eigenvalue）达到最大值时，$\boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^{\mathrm{T}}$达到最大值。换句话说，将数据降到一维就是求矩阵$\boldsymbol{C}$的最大特征向量，求得这个向量之后，根据公式（10-10）就可以得到降维后的数据。类似地可以证明，若要将数据降到k维，则需要求矩阵$\boldsymbol{C}$的前k个特征向量（相应的特征值从大到小排在前k位）。

从上面的推导过程可以看到，主成分分析的设计出发点是使降维后的数据保留尽可能多的信息。其实可以从另一个角度来考虑降维这个问题：如何使降维过程中的信息损失达到最小。沿用上面将数据降到一维的例子，由于对于第i点，降维后的数据为$\widetilde{\boldsymbol{X}}_i = (\boldsymbol{X}_i\boldsymbol{U}^{\mathrm{T}})\boldsymbol{U}$，那么损失掉的信息为$\boldsymbol{L}_i = \boldsymbol{X}_i - \widetilde{\boldsymbol{X}}_i$。从直观上来讲，原始数据到降维直线的距离就为损失的信息量。根据最小损失信息的原则，最佳降维向量的估算公式如下所示。

$$\widehat{\boldsymbol{U}} = \operatorname{argmin}_{\|\boldsymbol{U}\|=1} \sum_{i=1}^{n} \|\boldsymbol{L}_i\|^2 \tag{10-14}$$

如图 10-20 所示，数学上可以证明公式（10-14）和公式（10-13）的两个最优化问题是等价的，因此主成分分析在降维过程中同时达到了两个目的，一是尽可能地保留数据间的差异，二是尽可能地减少信息的损失。

[20] 若数据的中心不是原点，只需对数据减去这个中心就能满足正文中的假设。而且这样的数据变换并不影响降维的结果，因为数据的变换幅度并没有受影响。

书籍是人类进步的阶梯

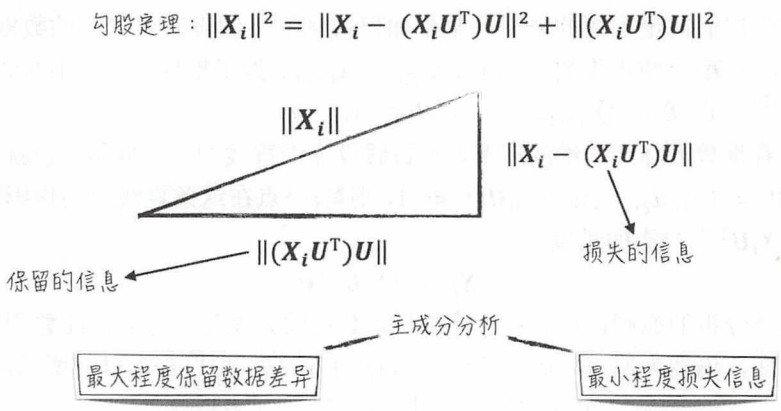


图 10-20

10.4.2 模型实现

主成分分析的代码实现也比较简单，如程序清单 10-4 所示。

（1）scikit-learn 使用 PCA 类来实现主成分分析，如第 1 行代码所示。

（2）在 PCA 类中，参数“n_components”表示降维后数学的维度，如第 7 行代码所示，这里的代码表示将数据降到二维。

（3）从 10.4.1 节的推导过程可以得到，主成分分析要求数据的中心是原点，对于中心不是原点的数据，需要先做数据平移（减去数据的中心点），再进行降维。scikit-learn 已经在 PCA 类的实现里考虑到了这一点，因此对于任何数据，都可以直接调用它的 fit 函数训练模型，如第 8 行代码所示。

程序清单 10-4 主成分分析

```
1  from sklearn.decomposition import PCA
2
3  def trainModel(data):
4      """
5      使用 PCA 对数据进行降维
6      """
7      model = PCA(n_components=2)
8      model.fit(data)
9      return model
```

根据上面的讨论，主成分分析可以把数据降到k维。与聚类算法里的聚类个数类似，我们需要选择最适合数据的k值。解决这个问题的方法与 k-means 中的方法几乎一模一样。具体地，首先需要定义评估降维效果的指标，由于主成分分析的目的是尽可能地保留数据的差异，因此，降维后数据方差占原始数据方差的比例是一个很自然的评估指标。

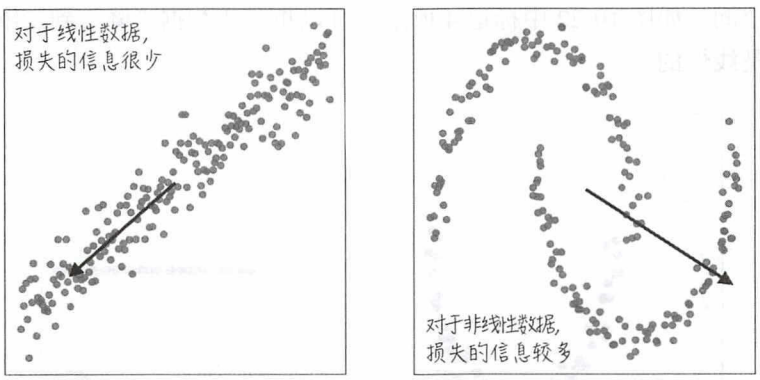
书籍是人类进步的阶梯

数学上可以证明，数据降维后，它在每一维度的方差与相应的特征值成正比（具体的细节请参考公式（10-12）和公式（10-13））。因此，只需将数据协方差矩阵（10.4.1 节里的矩阵C）的特征值从大到小排列，然后使用 10.1.3 节中讨论的 elbow method 选择最佳的k值。

最后需要注意的一点是，在主成分分析中，k的取值并不是任意的：它需要同时小于原始数据的维度m以及原始数据的个数n。

10.4.3 核函数

从直观上来讲，主成分分析是用不断画直线，并向直线做投影的方法来对数据进行降维的，因此它对线性数据的降维效果很好，也就是说降维过程中损失的信息（原始数据到降维直线的距离平方和）较小，如图 10-21a 所示，图中的箭头表示降维直线。但如果数据本身是非线性的，那么主成分分析的降维效果就比较差：降维过程中损失的信息较多，如图 10-21b 所示。因为这种情况下，数据并不是沿着某条直线变动，而是沿着一条弯曲的曲线上下浮动。



（a）　　（b）

图 10-21[21]

在第 8 章中讨论支持向量学习机（SVM）时，我们也曾碰到过类似的问题。简单回顾一下，经典的支持向量学习机（linear SVM）只能处理线性数据的分类问题。对于非线性数据，需要使用核函数将低维空间中的数据映射到高维空间，使得数据在高维空间里是近似线性的，这样就可以在后者的基础上，使用支持向量学习机解决问题。主成分分析对非线性数据的处理也是类似的，首先通过核函数将数据升到高维空间，然后再使用模型将高维空间里的数据降到所需的维度，这样的方法在学术上被称为 kernel PCA。kernel PCA 的目的是对数据进行降维，但它的第一步却是使用核函数对数据进行升维，这种做法虽然有点匪夷所思，甚至有

[21] 图 10-21 和图 10-22 的完整实现请参考随书配套的代码/ch10-unsupervised/dimension_reduction/kernel_pca.py。

点南辕北辙的意思，但这样却能有效地对非线性数据进行降维。

以图 10-22 为例，原始数据在平面上呈月牙形，如标记 1 所示。虽然数据里并没有标签变量，但为了更直观地展示降维的效果，我们用三角形表示其中一个月牙的数据，用圆形表示另一个月牙的数据。

- 如果直接使用主成分分析对数据进行降维，得到的结果并不理想，因为数据降维后，两个不同月牙的数据混杂在了一起，极大地损失了原始数据的信息，如图中标记 2 所示。值得注意的是，不使用核函数的主成分分析在学术上又被称为线性主成分分析（linear PCA），这样称呼的原因请参考 10.4.4 节。
- 如果在主成分分析的基础上加上 RBF 核函数（RBF kernel PCA），则降维的效果很好很多。如图 10-22 中标记 3 所示，数据降到一维后，两个月牙的数据是分开的，说明损失的信息相对较少。
- 由于使用了核函数将原始的二维数据映射到了高维空间，而后者才是真正进行降维运算的数据，因此，使用核函数主成分分析对数据进行降维时，能将数据“降”到比原始数据更高维度的空间。如图 10-22 中标记 4 所示，可以将月牙数据“降”到二维，在新的空间里，数据几乎是线性的。

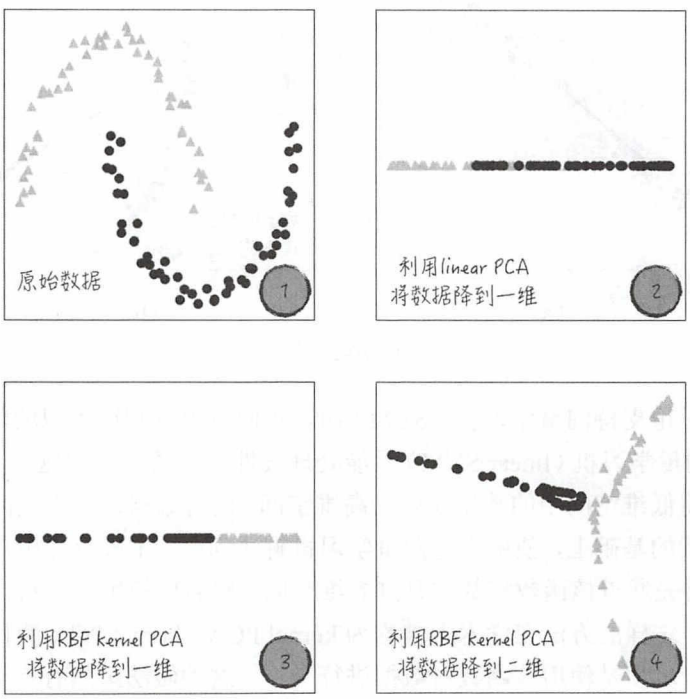


图 10-22

仅供非商业用途或交流学习使用

使用核函数的主成分分析虽然是一个不太容易理解的模型，在数学上的推导也很复杂，但它的代码实现依然十分简单，如程序清单 10-5 所示。

（1）scikit-learn 中的 KernelPCA 实现了核函数加主成分分析的模型组合，如第 1 行代码所示。

（2）KernelPCA 的调用方式和 PCA 是差不多，如第 7～9 行代码所示。其中比较重要的参数有两个：一个是“n_components”，它表示降维后的维度；另一个是“kernel”，它表示使用核函数的种类，这个参数的使用方法与支持向量学习机里的一模一样[22]，有关细节请参考 8.2.5 节。

程序清单 10-5 核函数+主成分分析

```
1  from sklearn.decomposition import KernelPCA
2
3  def trainKernelPCA(data):
4      """
5      使用带有核函数的主成分分析对数据进行降维
6      """
7      model = KernelPCA(n_components=2, kernel="rbf", gamma=25)
8      model.fit(data)
9      return model
```

10.4.4 Kernel PCA 的数学原理

上一节针对 kernel PCA 的讨论主要是从直观和实用层面出发的，但对主成分分析配合核函数使用的数学基础并没有涉及，本节将补上这一块。这部分内容虽然能帮助我们更好地理解 kernel PCA，但它太过于“数学”，十分抽象，对理论细节不感兴趣的读者可略过此节。

针对下面的数学讨论，我们沿用 10.4.1 节中的数学记号：假设需要降维的数据有n个，数据维度位为m维，记第i个数据为$\boldsymbol{X_i} = (x_{i,1}, x_{i,2}, \ldots, x_{i,m})$，而且这$n$个数据的中心为原点。

为了便于讨论，不妨假设需要将数据降到一维（降维向量用$\boldsymbol{U}$表示），那么根据 10.4.1 节中的讨论，主成分分析的降维目标和相应的最佳降维向量如公式（10-15）所示，其中$\boldsymbol{C}$为数据的协方差矩阵。

$$\begin{aligned} &\max_{\|\boldsymbol{U}\|=1} \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^{\mathrm{T}} \\ \widehat{\boldsymbol{U}} &= \operatorname{argmax}_{\|\boldsymbol{U}\|=1} \boldsymbol{U}\boldsymbol{C}\boldsymbol{U}^{\mathrm{T}} \end{aligned} \tag{10-15}$$

在数学上，利用拉格朗日对偶（具体细节请参考 8.2.2 节），可以将公式（10-15）中的$\widehat{\boldsymbol{U}}$表示为$\widehat{\boldsymbol{U}} = \sum_{i=1}^{n} \hat{\alpha}_i \boldsymbol{X_i}$。利用这个表达式，可以将公式（10-15）中主成分分析的降维目标改写为公式（10-16）的形式。

[22] 除了 kernel 外，有核函数相关的参数还有 gamma、coef0 和 degree。这些参数的含义和使用方法也和支持向量学习机中的一模一样。

仅供非商业用途或交流学习使用

$$\max_{\alpha L\alpha^{\mathrm{T}}=1} \boldsymbol{\alpha L L \alpha}^{\mathrm{T}}$$

$$\hat{\alpha} = \operatorname{argmax}_{\alpha L\alpha^{\mathrm{T}}=1} \boldsymbol{\alpha L L \alpha}^{\mathrm{T}} \tag{10-16}$$

在上面的公式中，$\boldsymbol{\alpha}$为n阶行向量；$\boldsymbol{L}$是一个$n \times n$的矩阵，相应的元素为$l_{i,j} = \boldsymbol{X}_i \boldsymbol{X}_j^{\mathrm{T}}$，在学术上$\boldsymbol{L}$被称为线性核矩阵（对应着 linear kernel），这也是主成分分析也被称为 linear PCA 的原因。

有了公式（10-16）作为基础，我们很容易就可以在主成分分析中引入核函数：只需将公式中的线性核矩阵L换成相应核函数的核矩阵$\boldsymbol{K}$即可。下面以 RBF 核函数为例，整个过程如图 10-23 所示。

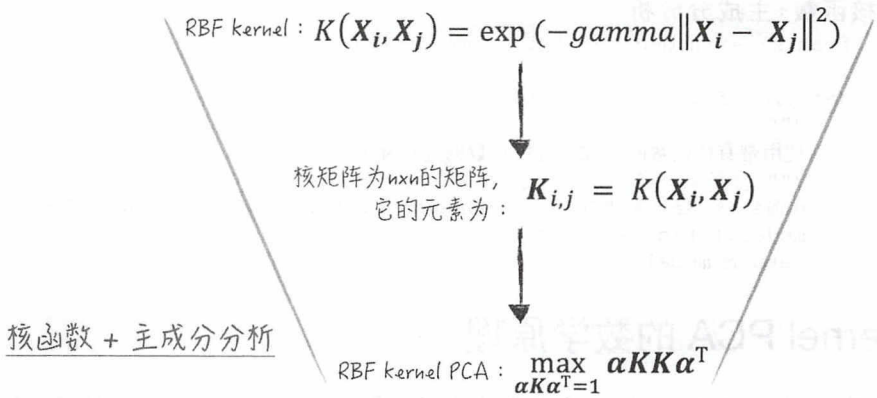


图 10-23

10.4.5 应用示例

主成分分析在实际中的应用十分广泛，这里举几个常见的例子。

- 由于我们生活的现实世界是一个三维空间，因此对于高于三维的数据，我们无法在视觉上直观地呈现它们。对于这类高维数据，数据科学家们通常会借助数学工具从侧面去了解它们的特性，但这样的分析结果难免有些生涩难懂。这种情况下，可以借助主成分分析法将数据降到二维或者三维，然后在坐标系里“画出”数据。这种方式可以清晰有效地传达信息，让洞察见解跃然纸上。
- 在监督式学习中，当训练数据的维度很多时，很容易引起过拟合（overfitting）的问题，因为数据维度高意味着模型的变量多，也就是所用模型的复杂度很高。这种情况下，使用主成分分析对训练数据的自变量进行降维能有效地解决过拟合的问题。当然降维还有另外两个好处：一是模型的工程实现难度降低，更容易得到正确的参数估计值；二是能有效地消除随机噪声因素的影响，更好地抓住数据的主要特征。

仅供非商业用途或交流学习使用

- 图像处理常常涉及很高维的数据，比如对于一个长边有m个像素点，宽边有n个像素点的图片，表示它的向量为$m \times n$维。处理这种高维数据是很困难的，我们常常使用主成分分析法将其降维，使图片信息浓缩到较低维度的向量。

10.5 奇异值分解

本节将继续讨论另一种很常见的数据降维模型——奇异值分解。这个模型不但能被用于降低数据的维度，而且还能从表示表象的数据中挖掘出内在的隐含因素，因此它在自然语言处理和推荐系统领域应用十分广泛。

10.5.1 定义

奇异值分解（Singular Value Decomposition，SVD）其实是一个纯粹的数学概念，它是一个有关矩阵分解的定理。下面是奇异值分解的具体内容（我们先暂时不管这个定理在机器学习中的应用）。不妨假设$\boldsymbol{A}$是一个$m \times n$的矩阵，数学上可以证明，矩阵$\boldsymbol{A}$能被分解为 3 个矩阵的乘积：

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}} \tag{10-17}$$

公式（10-17）中$\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}$的具体含义如下。

- $\boldsymbol{U}$是一个$m \times m$的正交矩阵（orthonormal matrix），也就是说这个矩阵和它转置矩阵的乘积为单位矩阵，即$\boldsymbol{U}\boldsymbol{U}^{\mathrm{T}} = \boldsymbol{U}^{\mathrm{T}}\boldsymbol{U} = \boldsymbol{I}$。
- $\boldsymbol{\Sigma}$是一个$m \times n$的对角矩阵，这个矩阵的对角线元素按从大到小排列，并且都是非负实数，这些对角线上的元素在学术上被称为奇异值（singular value）。现在将上面的语句翻译成数学式子：记$\boldsymbol{\Sigma}_{m \times n} = (\varphi_{i,j})$，则矩阵元素满足公式（10-18），其中$l = \min(m, n)$。

$$\varphi_{i,j} = \begin{cases} 0, & i \neq j \\ \geqslant 0, & i = j \end{cases}; \ \varphi_{1,1} \geqslant \varphi_{2,2} \geqslant \ldots \geqslant \varphi_{l,l} \geqslant 0 \tag{10-18}$$

- $\boldsymbol{V}$也是一个正交矩阵，它的维度是$n \times n$。

10.5.2 截断奇异值分解

奇异值分解虽然在数学上比较完美，但在机器学习实践中，几乎没有应用场景需要计算完整的奇异值分解，因为我们关注的往往是最大的几个奇异值。为此学术界引入了另一种矩阵分解方法——截断奇异值分解（truncated SVD）。

对于矩阵$\boldsymbol{A}$，通过奇异值分解可以得到$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$，选取$\boldsymbol{\Sigma}$中最大的$k$个奇异值，得到新的$k \times k$对角线矩阵$\boldsymbol{\Sigma}_k$；选取矩阵$\boldsymbol{U}$中相应的前$k$列，得到$m \times k$的矩阵$\boldsymbol{U}_k$；类似地选取矩

仅供非商业用途或交流学习使用

阵$\boldsymbol{V}$中前k列，得到$n \times k$的矩阵$\boldsymbol{V}_k$。上述的$\boldsymbol{\Sigma}_k, \boldsymbol{U}_k, \boldsymbol{V}_k$这 3 个矩阵就是截断奇异值分解的结果，整个过程如图 10-24 所示。

$$\text{truncated SVD}: \boldsymbol{A} \approx \boldsymbol{U}_k \boldsymbol{\Sigma}_k \boldsymbol{V}_k^{\mathrm{T}} \qquad \text{SVD}: \boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathrm{T}}$$

$$\underset{\boldsymbol{A}}{\begin{pmatrix} 0, & 1, & 2, & 3 \\ 4, & 5, & 6, & 7 \\ 8, & 9, & 10, & 11 \end{pmatrix}} = \underset{\boldsymbol{U}}{\begin{pmatrix} \overset{\boldsymbol{U_1}}{\boxed{\begin{matrix} -0.15, \\ -0.5, \\ -0.85, \end{matrix}}} & \begin{matrix} -0.9, & 0.41 \\ -0.29, & -0.81, \\ 0.32, & 0.41 \end{matrix} \end{pmatrix}} \times \underset{\boldsymbol{\Sigma}}{\begin{pmatrix} \overset{\boldsymbol{\Sigma_1}}{\boxed{22.4,}} & 0, & 0 & 0 \\ 0, & 1.96, & 0, & 0 \\ 0, & 0, & 0, & 0 \\ 0, & 0, & 0, & 0 \end{pmatrix}} \times \underset{\boldsymbol{V}}{\begin{pmatrix} \overset{\boldsymbol{V_1}}{\boxed{\begin{matrix} -0.39, \\ -0.46, \\ -0.53, \\ -0.59, \end{matrix}}} & \begin{matrix} 0.74, & 0.32 & -0.44 \\ 0.29, & -0.10, & 0.83 \\ -0.15, & -0.77, & -0.33 \\ -0.59, & 0.54, & -0.06 \end{matrix} \end{pmatrix}^{\mathrm{T}}}$$

图 10-24

值得注意的是，在截断奇异值分解中的k值与主成分分析中的降维维度是类似的，可以在一定范围内任意选择。它的选择方法依赖于具体的应用场景，通常使用网点搜寻（grid search）的方法来找到最佳的取值。

以上就是截断奇异值分解的理论部分，现在来讨论两个经典的应用场景：潜在语义分解和大型推荐系统。

10.5.3 潜在语义分析

潜在语义分析（Latent Semantic Analysis，LSA）是自然语言处理领域一个很常见的应用。这个模型将根据训练文本，将文字和相应的文章转换为向量，并使得这些向量能最大程度地代表文字或文章的语义。用数学语言解释就是，在比较理想的情况下，潜在语义分析将把同义字词转换为夹脚很小的向量（向量的方向差不多是一致的），把反义词转换为夹脚很大的向量。这样就可以很神奇地把人类才能理解的语义转换为模型和机器能够处理的向量，在这些向量的基础上可以衍生出很多应用，比如文本的分类、情感分析等。那么潜在语义分析是如何进行的呢？它与截断奇异值分解有什么样的关系呢？这些内容是下面讨论的重点。

先反过来考虑语义分析的问题，假设我们可以用k维向量去同时表示文章中出现的文字[23]以及文章本身。向量的每一维代表一个语义主题，比如表达正面情感的程度、表达意思的复杂程度等，而一个文字或文章在某一维的数值表示它在这个语义主题上的权重。那么如果一个文字和一篇文章在语义上很接近，则它们在各个语义主题上的权重是很相似的，对应到数学上就是相应的向量夹脚很小（当然这种情况下，两个向量的内积是很大的）。

另一方面，从人类语言的角度来讲，如果一个文字和一篇文章在语义上的契合度很高，

[23] 本节中提到的文字指的并不是中文里的汉字，而是语义单元，其含义更接近于日常生活中的词，这与 9.2.1 节中的设定是类似的。

仅供非商业用途或交流学习使用

那么这个文字在这篇文章中的权重也就越大，比如在一篇爱情小说中总免不了出现大量的“爱”和“喜欢”。那么结合上面有关向量夹脚的讨论，文字向量和文章向量之间的内积应该很接近于该文字在这个文章中的权重。

因此，如果将文章的向量（行向量）放在一起组成矩阵$\boldsymbol{D}$；将文字的向量（列向量）放在一起组成矩阵$\boldsymbol{T}$；那么文字在文章中的权重矩阵$\boldsymbol{A}$将近似地等于矩阵$\boldsymbol{D}$和$\boldsymbol{T}$的乘积，如图10-25所示。$\boldsymbol{A}$为$m \times n$的矩阵（这代表了训练文本集中有m篇文章，而出现的文字有n个），它的每一行表示一篇文章，每一列表示一个文字，相应的元素$a_{i,j}$表示第j个文字在第i篇文章中的权重；矩阵$\boldsymbol{D}$的每一行表示表示一篇文章，而矩阵$\boldsymbol{T}$的每一列表示一个文字。

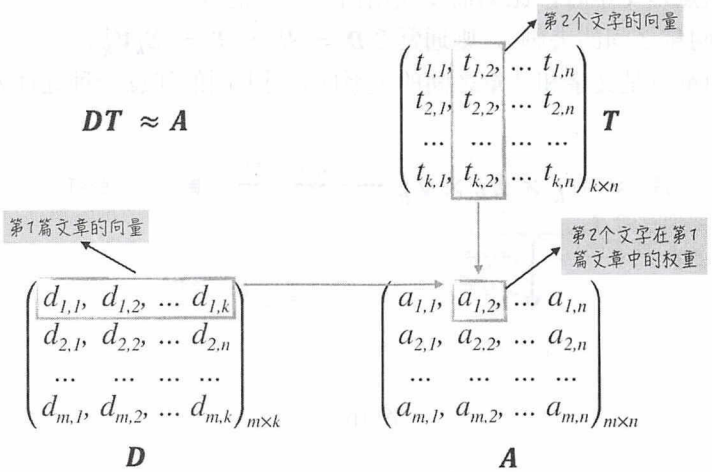


图 10-25

将上面的分析和截断奇异值分解结合起来会发现，如果使用k个奇异值分解矩阵$\boldsymbol{A}$，可以得到结果：$\boldsymbol{A} \approx \boldsymbol{U_k}\boldsymbol{\Sigma_k}\boldsymbol{V_k^{\mathrm{T}}}$，其中$\boldsymbol{U_k}$的形状与$\boldsymbol{D}$相似，两者都是$m \times k$的矩阵；而$\boldsymbol{V_k^{\mathrm{T}}}$的形状与$\boldsymbol{T}$相似，两者都是$k \times n$的矩阵，它们之间显然存在某种联系。由此看来通过截断奇异值分解，我们能从表面的文字权重矩阵，倒推出文字的向量表示和文章的向量表示，这就是潜在语义分析。具体的步骤如下。

（1）首先，对于给定的训练文本集，定义相应的文字权重矩阵$\boldsymbol{A}$。最常见的定义方式是9.2.4节里讨论的TF-IDF（Term Frequency -Inverse Document Frequency）。

（2）然后选择特定的k值，对矩阵$\boldsymbol{A}$使用截断奇异值分解，得到3个分解矩阵，它们相应的解释如下。

- $\boldsymbol{U_k}$的行代表文章，列代表语义主题，表示的是文章与各个语义主题的相关关系。但比较遗憾的是，我们并不知道这k个语义主题具体代表着什么，只知道它们能在最大程度上代表文章的含义 。这其实是潜在语义分析的缺点之一：模型结果不好或者说无法解释。

仅供非商业用途或交流学习使用

- $\boldsymbol{V_k}$是表示文字的矩阵，行代表文字，列代表语义主题，表示文字与语义主题的相关关系。
- $\boldsymbol{\Sigma_k}$是对角矩阵，它在对角线上的元素表示相应语义主题的重要程度，也就是说这个语义主题能在多大程度上表示训练文本中文字权重的变化。

（3）最后是根据具体的应用场景，构建相应的文字矩阵$\boldsymbol{T}$和文章矩阵$\boldsymbol{D}$，常用的处理方法如图 10-26 所示。

- 当建模重点是文章时，比如需要对文本进行分类（从模型角度来看，就是要对文章矩阵中的行向量进行聚类），则通常令$\boldsymbol{D} = \boldsymbol{U_k}\boldsymbol{\Sigma_k}$和$\boldsymbol{T} = \boldsymbol{V_k^T}$。
- 当建模重点是文字时，比如需要找出给定文字的同义词（从模型上来看，就是要计算文字矩阵中列向量之间的夹脚），则通常令$\boldsymbol{D} = \boldsymbol{U_k}$和$\boldsymbol{T} = \boldsymbol{\Sigma_k}\boldsymbol{V_k^T}$。
- 当研究的重点是文字和文章之间的关系时，则上面的任意一种处理方法都是合适的。

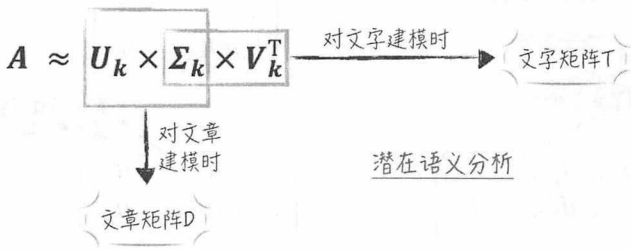


图 10-26

scikit-learn 对截断奇异值分解的实现是 sklearn.decomposition.TruncatedSVD，在这个类的基础上，可以较容易地完成语义的分析，比如使用 Pipeline 将 TruncatedSVD 和 k-means 联结起来解决文本分类的问题。限于篇幅，具体细节请参考 scikit-learn 官网，在此不再赘述。

10.5.4 大型推荐系统

大型推荐系统在互联网上的应用非常广泛，比如在网上购物时，电商平台会根据你的个人信息和历史购买行为向你推荐商品；又比如使用手机收听音乐时，相应的 App 会根据你的喜好，推送歌曲。从模型的角度来看，这些推荐行为其实对应着一个分类问题。

- 站在用户的角度，推荐系统将所有物品分为两类，一类是这个用户感兴趣的，另一类是这个用户不感兴趣的。
- 反过来，站在物品的角度，推荐系统将用户分为两类，对该物品感兴趣的和对该物品不感兴趣的。

针对这样的分类问题，之前章节讨论的很多模型都能有效地解决，比如逻辑回归、支持向量学习机等。但这些模型都只能对一个角度进行建模，比如从用户的角度出发，则建模数

仅供非商业用途或交流学习使用

据是物品的特征，而用户的喜好倾向由模型参数体现，也就是说，这时需要对每个用户单独进行建模。如果从物品的角度出发，也可以得到类似的结论。但问题是，上面的互联网应用场景有一个共同的特点就是用户数量和物品数量都很大，这导致经典的分类模型无法使用（否则需要同时搭建和训练几万，甚至几百万的模型）。

为了解决这个问题，学术界提出了新的模型，而这个模型与 10.5.3 节中讨论的潜在语义分析非常相似。通过某种矩阵分解的方法，用k个特征（k维向量）来同时表示用户和物品。在这个矩阵分解的基础上，用户对物品感兴趣的程度就可以用相应的向量内积来表示了。下面以音乐推荐为例子，说明模型的具体步骤。

在这个场景中，我们能收集到用户对部分音乐的评分[24]，可以使用所谓的偏好矩阵$\boldsymbol{M}$（preference matrix）来记录这些评分：它是一个$m \times n$的矩阵，其中行向量表示用户，列向量表示音乐，相应的元素等于用户对音乐的评分，如图 10-27 所示。值得注意的是，偏好矩阵$\boldsymbol{M}$是不完整的，里面大部分元素的值是未知的（在现实生活中，一个用户只会对很少一部分音乐进行评分），这与 10.5.3 节中的矩阵$\boldsymbol{A}$是完全不同的，后者的每一个元素都是已知的。事实上，我们建模的目的就是要将偏好矩阵$\boldsymbol{M}$“填满”（使用模型的预测值），因为这相当于已知了用户的喜好，在此基础上为用户推荐音乐就是件很轻松的事情了。

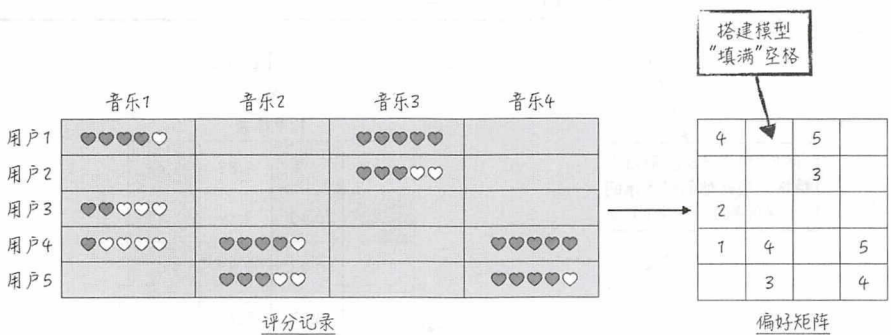


图 10-27

从数学上来讲，模型对偏好矩阵$\boldsymbol{M}$的处理思路与潜在语义分析是一致的，按公式（10-19）的方式将$\boldsymbol{M}$分解成两个矩阵的乘积，其中$\boldsymbol{X}$为$m \times k$的矩阵（k为模型的一个参数），它的行向量表示用户，每一列对应着一个隐含特征；$\boldsymbol{Y}$是$n \times k$的矩阵，它的行向量表示音乐，每一列的含义与$\boldsymbol{X}$中的列是一样的。

[24] 在很多实际的应用场景里，我们并得不到客户的直接反馈。相反，我们能得到一些间接的反馈，比如页面浏览次数、商品购买次数、视频播放次数等。在这些间接反馈的基础上，可以通过某种转换，定义出相应的偏好矩阵。对相关细节感兴趣的读者可参考 Hu Y, Koren Y, Volinsky C. Collaborative Filtering for Implicit Feedback Datasets[C]// Eighth IEEE International Conference on Data Mining. IEEE, 2009:263-272.

仅供非商业用途或交流学习使用

$$\boldsymbol{M} \approx \boldsymbol{X}\boldsymbol{Y}^{\mathrm{T}} \tag{10-19}$$

由于$\boldsymbol{M}$并不是一个完整的矩阵，因此无法使用之前讨论的截断奇异值分解来估算矩阵$\boldsymbol{X},\boldsymbol{Y}$。因此为了求解这个问题，我们定义如下的损失函数，其中，$m_{i,j}$表示矩阵$\boldsymbol{M}$中已经定义的元素；X_i是矩阵$\boldsymbol{X}$的行向量，Y_j是矩阵$\boldsymbol{Y}$的行向量，α为惩罚项权重。

$$L = \sum_{i,j}(m_{i,j} - \boldsymbol{X}_i\boldsymbol{Y}_j^{\mathrm{T}})^2 + \alpha(\|\boldsymbol{X}_i\|^2 + \|\boldsymbol{Y}_j\|^2) \tag{10-20}$$

与其他机器学习模型一样，矩阵$\boldsymbol{X},\boldsymbol{Y}$的估算原则是使得损失函数达到最小值。如果抛开惩罚项，公式（10-20）表示，模型希望预测值$\boldsymbol{X}_i\boldsymbol{Y}_j^{\mathrm{T}}$能尽可能地逼近观测到的实际值$m_{i,j}$。

另一方面，如果得到了矩阵$\boldsymbol{X},\boldsymbol{Y}$的估计值，那么可以得到$\boldsymbol{M}$整个矩阵的预测值。以图 10-27 中的数据为例，模型估算的整个过程如图 10-28 所示。

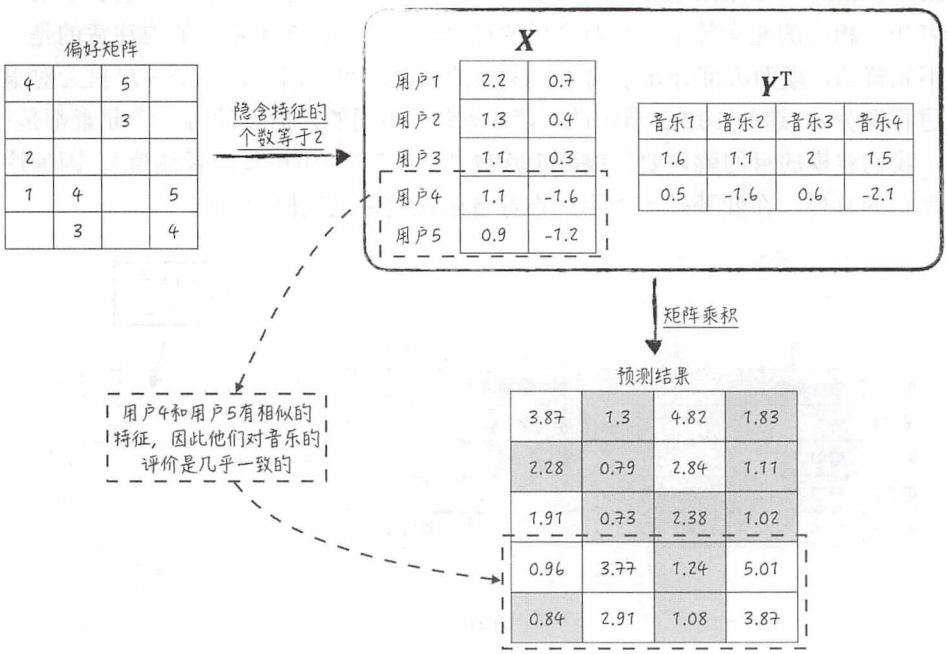


图 10-28

这个模型虽然在数学上处理比较复杂，但它做推荐的出发点却是很朴素的。如果两个用户有相似的评分行为，那么他们的喜欢也是大致相同的。于是可以根据其中一个人的评分行为对另一个人做预测。例如图 10-28 中的例子，用户 4 与用户 5 的评分记录大体一致，又因为用户 4 对音乐 1 的评分只有 1，那么可以推测用户 5 对音乐 1 的评分也将很低。这种建模思路在学术上被称为协同过滤（collaborative filtering），而上面的讨论的矩阵分解只是从数学上量化地实现了这个思路。

虽然上述的模型可以处理大量用户和大量物品并存的推荐问题，但在实际生产中，很少会直接使用它进行推荐主要有如下两个原因。

- 在很多应用场景下，需要处理的用户数和物品数实在太多，即使这样专门设计的模型也难以承受。
- 在实际中，直接使用此模型的效果也不一定好，因为这个模型只考虑用户对物品的偏好，其他对推荐有用的信息，比如用户的个人信息、物品的属性和用户行为特征等，这个模型都无法使用。

为了更好地推荐效果，通常会将这个模型和其他模型联结起来使用，比如先用上述的模型预测用户对某一类物品的偏好，再在此基础上，使用其他模型得到最终的预测结果。这部分内容的细节超出了本书的范围，在此就不做深入讨论了。

从工程实现上来讲，上面讨论的模型通常会处理非常庞大的数据集，因此需要在分布式计算框架下实现模型，这样就可以利用集群的力量来训练模型。对于分布式机器学习（或者大数据机器学习），第 11 章将介绍一款很优秀的开源算法库 Spark ML。它里面就有上述模型的实现，具体的类为 pyspark.ml.recommendation.ALS。代码层面的具体细节请参考第 11 章和 Spark 官网。

10.6 本章小结

本章是比较特殊的一章，讨论的重点是非监督式学习，也就是在没有标签变量的情况下，为数据搭建模型。这样的模型通常分为两类：聚类模型和降维模型。

对于聚类，这一章主要讨论了 3 种经典的模型：k-means、混合高斯以及谱聚类。

k-means 是非常简单和直观的聚类模型，它用欧式距离来度量数据间的相似度。用类似“画圆圈”的方法来完成聚类运算。

由于 k-means 对相似度的定义，它隐含着的假设是数据的各个维度是均质的，也就是数据在各个维度的变化幅度是差不多的，这使得模型的适用范围很窄。为了解决这个问题，我们讨论了混合高斯模型，它是一个生成式模型，是二次判别分析的非监督版本。混合高斯对聚类的处理可以形象地理解为“画椭圆”，因此，k-means 其实是这个模型的一个特例。

虽然混合高斯的适用范围更广，但它依然没办法对类似流体一样的数据做聚类，比如二维空间里月牙形状的数据。而谱聚类却很擅长处理这类数据的聚类问题。谱聚类的思路是先将数据转换为连通图，再借助 Laplacian matrix 的特征向量完成连通图的向量化，最后在向量的基础上完成聚类运算。

这 3 种聚类模型虽然差别很大，但它们都面临着同一个问题，就是如何选择聚类个数k

（事实上，这是所有聚类模型都面临的问题）。对于这个问题，并没有特别一个通用的解决方法。本章结合具体的模型，讨论了几种比较常用的选择策略，比如针对 k-means 的 elbow method 以及针对混合高斯的 BIC。

对于降维，本章讨论了两种模型：主成分分析和截断奇异值分解。从建模思路上来看，主成分分析将在降维过程中尽可能地保留数据间的差异，而截断奇异值分解的目的是尽可能地还原原有矩阵。这两种模型虽然在表面上区别很大，但它们的理论基础都是矩阵的特征向量，工程实现上也有很多相通之处。从使用的角度来看，主成分分析主要用于数据可视化和降低数据中随机因素的干扰，而截断奇异值分解则主要用于生成更高效的向量来表示数据。

除了本章讨论的这些模型，非监督式学习里还有很多其他模型，但限于篇幅，本书就没办法一一讨论了。想要了解更多内容的读者可参考其他书籍，比如 Trevor Hastie 等人编著的 *The Elements of Statistical Learning*。

到本章为止，有关机器学习领域里最经典的内容，我们已经讨论得差不多了。接下来的章节将重点讨论数学科学里的前沿领域。第 11 章将介绍分布式机器学习：如何在大数据上搭建以及训练模型；第 12、13 章将讨论神经网络以及深度学习：如何通过模型联结，模拟人的大脑。

第11章

分布式机器学习：集体力量

二人同心，其利断金；同心之言，其臭如兰。

——《周易》

11.1 Spark 简介

11.2 最优化问题的分布式解法

11.3 大数据模型的两个维度

11.4 开源工具的另一面

11.5 本章小结

毫无疑问，我们生活在大数据时代：语言可以变成数据，图片可以变成数据，人的行为也可以变成数据，世间万物的信息皆可量化、数据化。而且随着互联网以及智能手机的普及，这些数据的收集可能变得更加廉价。这种现状和发展趋势也引发了一系列问题，比如数据的安全、个人隐私的保护等，但对数据科学这门学科而言，这无疑是一件大喜事。从实用的角度来讲，数据越多，意味着可能的应用场景也就越多，由模型挖掘出来的数据价值也就越高。从技术的角度来讲，大数据至少能从两个方面带来好处。第一，在数据量较大的情况下，模型过拟合（overfitting）的问题就不容易发生，因为相比于数据量，模型参数的个数就显得微不足道了，即使是非常复杂的模型。第二，训练数据集较大能有效地提高模型参数的稳定性。以第 4 章中讨论的线性回归模型为例，模型参数的方差与训练数据集的大小成反比。

大数据在带来红利的同时，也从数学层面和工程实现层面上给数据科学提出了巨大的挑战。

在数学层面上，我们将面对比以前多得多的变量，而这些变量并不都与被预测量有相关关系。如果不加筛选，一股脑地将所有变量都“塞进”模型，那么往往会导致错误的结论。有关这方面的问题，我们在第 4、5、7 章里已经详细讨论过了。

在工程实现层面上，在实际的生成环境中，收集到的数据量已经远远超出了单台计算机的处理能力，无论是存储能力还是计算能力。而且在可预见的未来，数据的增长速度仍会大幅超过计算机性能进步的速度。因此，在处理大数据时，需要将很多台计算机联结起来组成一个巨大的分布式计算集群，将收集到的数据分散地存储在这些机器上，如图 11-1 所示。

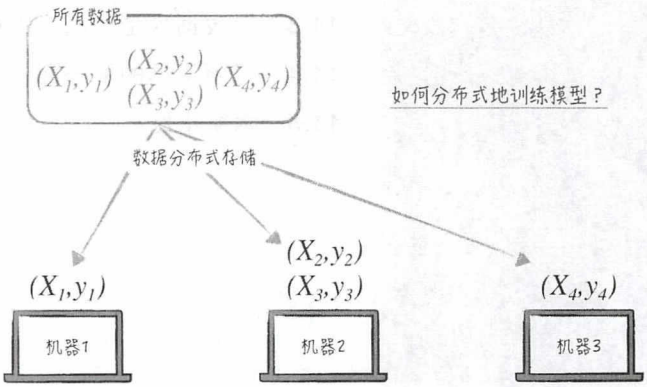


图 11-1

这样的架构虽然解决了数据的存储问题，但给数据的计算增加了难度。对于有些简单计算，增加的难度并不大，比如现在需要统计数据里一共有多少条记录，那么只需让每台机器先独立地对局部数据做统计，再对各台机器上的结果求和即可。

但对于复杂的模型训练，问题就没这么简单了。以第 4 章中讨论的线性回归模型为例，不妨假设训练模型的数据被记为$\{(\boldsymbol{X}_i, y_i)\}$，则模型的损失函数为$L = \sum_i (y_i - \boldsymbol{X}_i\boldsymbol{\beta})^2$。在单

机版本的算法中，模型的参数由第 6 章里讨论的随机梯度下降法[1]来估算。在估算的过程中，需要用到所有的训练数据。但现在这些数据分布在不同的机器上，并且其数量大到无法将所有数据放到一台机器上，因此单机版本的参数估计算法在这里就无法使用了。

更困难的是，模型的损失函数L并不像上面提到的求和运算，能通过简单的拆分将其转换为只需使用局部数据的子问题来解决：如果在每台机器上独立地运行梯度下降法，是可以得到一系列“临时”的参数估计值，但根据这些“临时”的估计值，我们无法得到正确的最终结果（参数估计值）。由此可知，机器学习的分布式实现并不是那么直接，需要更加精细的分析和设计。这也是本章的讨论重点。

为了避免陷入空谈的局面，在讨论分布式机器学习的理论细节之前，先简单介绍一下 Apache Spark 这个很常用同时也很优秀的分布式计算工具。

11.1 Spark 简介

Apache Spark 是一款开源的分布式计算引擎，它诞生于 2010 年。虽然这个项目的时间不长，但它已经在大数据并行计算领域独占鳌头，成为了业界大数据处理的标准配置。而且由于其强大的生态系统（与 Spark 配套使用的其他 IT 工具），在可预见的未来，Spark 都将是这个领域的主流工具。

与前面章节一直讨论的 Python 不同，Spark 并不是一种编程语言。事实上它是由 Scala[2]开发的第三方工具包，有点类似于 Python 里面的第三方库，比如之前章节涉及的 pandas 和 scikit-learn。基于 Spark，我们可以很方便地在分布式系统里清洗数据、提取特征以及搭建模型。

虽然 Spark 的核心部件并不是由 Python 开发的，但是这个项目提供非常成熟的 Python API，方便熟悉 Python 的数据科学家使用 Spark，这也正是本章讨论的重点（Spark 的 Python API 有个专有名字叫 PySpark）。事实上，除了 Python API，Spark 还提供 Java API 以及 R API，这是这个开源项目商业上非常成功的一点，因为提供多语言接口能有效地扩大潜在用户群，巩固 Spark 在大数据领域的领先地位（或者说垄断地位）。

Spark 涉及的内容很多，几乎包含了大数据的方方面面，比如流式实时计算（Spark

[1] 模型参数的估计算法有很多，但这些算法无一例外都需要使用全部的训练数据来计算参数的估计值。从数学意义上来讲，训练数据对模型的影响就是通过模型参数估计来体现的，因此不管估计算法的具体形式如何，它必然需要使用全体训练数据。

[2] Scala 是一种编程语言，它的设计初衷是集成面向对象编程和函数式编程的各种特性。Scala 运行在 Java 平台上，并兼容现有的 Java 程序，但与 Java 相比，它的语法结构更为灵活，非常适合机器学习的模型开发。

streaming）、ETL（Spark SQL）、机器学习（Spark ML）等。由于本书是一本讨论数据科学模型的书籍，因此针对 Spark 的讨论重点是如何利用它进行分布式机器学习，也就是 Spark ML。作为讨论的基础，本节将先介绍如何使用 Spark，包括它的安装指南、理论框架以及编程基础。

11.1.1 Spark 安装

正如上面的介绍，Spark 是由 Scala 开发的，因此需要首先安装 Java 和 Scala。值得注意的是，由于 Scala 的版本繁多，而且各个版本之间并不向下兼容，因此 Spark 只能运行在特定版本的 Scala 上。在本书编写时，Spark 只能运行在 Scala 2.10 和 Scala 2.11 这个版本上，所以本章将使用 Scala 2.11，而 Spark 的版本为 2.1.0。

1. 在 Windows 环境下安装 Spark

Windows 系统对 Spark 的支持并不好，经常会遇到一些很难解决的 bug，因此作者并不推荐在 Windows 下使用 Spark。

（1）从网上下载 Java 1.8 的安装包并运行它[3]，如图 11-2 所示。

下载Java安装包

Java SE Development Kit 8u144

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.89 MB	jdk-8u144-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.83 MB	jdk-8u144-linux-arm64-vfp-hflt.tar.gz
Linux x86	164.65 MB	jdk-8u144-linux-i586.rpm
Linux x86	179.44 MB	jdk-8u144-linux-i586.tar.gz
Linux x64	162.1 MB	jdk-8u144-linux-x64.rpm
Linux x64	176.92 MB	jdk-8u144-linux-x64.tar.gz
Mac OS X	226.6 MB	jdk-8u144-macosx-x64.dmg
Solaris SPARC 64-bit	139.87 MB	jdk-8u144-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.18 MB	jdk-8u144-solaris-sparcv9.tar.gz
Solaris x64	140.51 MB	jdk-8u144-solaris-x64.tar.Z
Solaris x64	96.99 MB	jdk-8u144-solaris-x64.tar.gz
Windows x86	190.94 MB	jdk-8u144-windows-i586.exe
Windows x64	197.78 MB	jdk-8u144-windows-x64.exe

64位系统
(较常见系统)

32位系统

图 11-2

（2）为了让 Java 能在 Windows 平台上运行，需要修改 Windows 的系统路径。如图 11-3 所示，进入“控制面板”搜索“环境”两个字，并单击“编辑系统环境变量”。

[3] 具体网址和步骤请参考随书配套的代码/ch11-spark/spark_install/Spark 安装补充说明.html。

在“系统变量”的界面上新建环境变量“JAVA_HOME”，它的变量值为 Java 的安装路径，比如这个例子中 Java 的安装路径为“C:\Program Files\Java\jdk1.8.0_144”。

接着新建环境变量“CLASSPATH”，这个环境变量用于定义 jar 包的存储路径，相应的值为“.;%JAVA_HOME%\lib\dt.jar;%JAVA_HOME%\lib\tools.jar”。注意不要忘记写上前面的字符“.”。

最后修改已有的环境变量“Path”，这个环境变量用于定义各种命令的存储路径。我们需要在已有值后面加上“;%JAVA_HOME%\bin;%JAVA_HOME%\jre\bin”，注意不要忘记写前面的字符“;”。

图 11-3

安装完成之后，如程序清单 11-1 所示验证安装是否成功（在命令提示符里运行，有关命令提示符的具体细节请参考 2.2.1 节）。

程序清单 11-1 在 Windows 上安装 Java

```
C:\Users\you> java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.7.0_67-b01)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

（3）在 Windows 下安装 Scala。从网上下载 2.11 版本的 Scala 安装包，在默认情况下，

安装成功后，系统会自动地修改相应的环境变量；如果没有安装成功（在命令提示符里运行命令“scala”报错），则和 Java 安装类似，需要手动修改环境变量。

具体来说，首先新建环境变量“SCALA_HOME”，它的变量值为 Scala 的安装路径，默认情况下为“C:\Program Files (x86)\scala”。

修改环境变量“CLASSPATH”：由于安装 Java 时已经创建了这个环境变量，因此只需在其后增加“.;%SCALA_HOME%\lib\dt.jar;%SCALA_HOME%\lib\tools.jar”。

修改环境变量“Path”：与 Java 安装类似，在此变量已有值的后面增加“;%SCALA_HOME%\bin;%SCALA_HOME%\jre\bin”。

在安装完成之后，运行如程序清单 11-2 所示的命令，验证安装是否成功。

程序清单 11-2 在 Windows 上安装 Scala

```
1 | C:\Users\you> scala -version
2 | Scala code runner version 2.11.11 -- Copyright 2002-2017, LAMP/EPFL
```

（4）由于 Windows 的文件系统和 Spark 支持的文件系统有冲突，因此必须下载额外的工具包。具体来说，首先在 C 盘创建目录“C:\hadoop\bin”，然后将随书配套的“winutils.exe”复制到该目录下，比如在本章中，选择的配套文件是“/ch11-spark/spark_install/winutils/hadoop-2.7.1/bin/winutils.exe”。

同之前类似，修改 Windows 的环境变量：创建“HADOOP_HOME”，它的变量值为“C:\hadoop”。

修改环境变量“Path”：在已有值的后面加上“;%HADOOP_HOME%\bin”。

（5）由于 Spark 在默认情况下集成了 HIVE[4]，因此需要特别为其创建路径，具体如下：首先创建路径“C:\tmp\hive”，然后运行下面的命令修改这个路径的权限，方便 Spark 访问：“winutils.exe chmod -R 777 C:\tmp\hive”。

（6）最后一步是下载并安装 Spark。这一步比较简单，只需从 Spark 官方网站下载已经编译好的压缩包，并解压即可。需要注意的是，下载的加压包应和第 4 步中选择的 Hadoop 版本保持一致，比如本章选择的就是“Apache Hadoop 2.7 and later”。程序清单 11-3 是启动 Spark 的命令。

程序清单 11-3 在 Windows 上安装 Spark

```
1 | [首先进入 Spark 的安装路径，也就是解压缩之后文件所在路径]
```

[4] Apache Hadoop 是一款开源的分布式大数据处理系统，它主要有两个不同功能的部件：一个是负责存储的分布式文件系统 HDFS，另一个是负责运算的计算框架 MapReduce。由于直接使用 MapReduce 操作数据比较困难，因此 Hadoop 开发了模仿传统的关系型数据库，开发了类 SQL 的数据操作工具 HIVE。

当然这些内容和 Spark 的使用都没有直接的联系，只是在实际生产中，Spark 常配合这些工具使用。所以在默认情况下，Spark 要求系统和这些工具是兼容的。

```
C:\Users\you>  you$ .\bin\pyspark --master local[*]
[在本地启动 Spark，可以得到如下的画面]
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Python version 3.5.3 (default, Feb 22 2017 21:28:42)
SparkSession available as 'spark'.
```

2. 在 Mac 环境下安装 Spark

（1）在 Mac 上安装 Java 和 Scala 就方便很多了。在第 2 章中安装 Python 时，我们安装了 macOS 缺失软件包管理器 homebrew（这个工具的安装指南请参考 2.2.2 节）。首先借助这个工具安装 Java，如程序清单 11-4 所示。

程序清单 11-4 在 Mac 上安装 Java

```
[localhost:~] you$ brew update
[localhost:~] you$ brew tap caskroom/cask
[更新 homebrew 的信息]
[localhost:~] you$ brew cask install java
[安装 Java 1.8]
[同时按下 Command 和 N 打开一个新的终端窗口，验证安装成功]
[localhost:~] you$ java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.8.0_144-b01)
Java HotSpot(TM) 64-Bit Server VM (build 25.144-b01, mixed mode)
```

（2）使用 homebrew 安装 Scala，如程序清单 11-5 所示。

程序清单 11-5 在 Mac 上安装 Scala

```
[localhost:~] you$ brew install homebrew/versions/scala211
[安装 Scala 2.11，如果已经安装了其他版本的 Scala，请使用下面的命令解除之前的安装]
[localhost:~] you$ brew unlink scala
[localhost:~] you$ brew link scala211 --force
[将系统的 Scala 路径指定到 Scala 2.11]
[同时按下 Command 和 N 打开一个新的终端窗口，验证安装成功]
[localhost:~] you$ scala -version
Scala code runner version 2.11.11 -- Copyright 2002-2017, LAMP/EPFL
```

（3）在 Mac 上安装 Spark。安装好 Java 和 Scala 后，Spark 的安装就很简单了，只需从 Spark 官方网站下载已经编译好的压缩包，并解压即可。程序清单 11-6 是启动 Spark 的命令。

程序清单 11-6 在 Mac 上安装 Spark

```
[首先进入 Spark 的安装路径，也就是解压缩之后文件所在路径]
[localhost:~] you$ ./bin/pyspark --master local[*]
```

书籍是人类进步的阶梯

```
[在本地启动 Spark，可以得到如下的画面]
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0
      /_/

Using Python version 2.7.9 (v2.7.9:648dcafa7e5f, Dec 10 2014 10:10:46)
SparkSession available as 'spark'.
```

3. 在 Linux 环境下安装 Spark

Linux 的版本很多，限于篇幅，这里只讨论 Ubuntu 上的安装步骤，而且下面的命令只在 Ubuntu 14.04 或者更高版本上测试过。

（1）Ubuntu 对 Java 和 Scala 的支持也比较友好，运行程序清单 11-7 便可安装 Java 1.8。

程序清单 11-7 在 Ubuntu 上安装 Java

```
you@you:~$ sudo add-apt-repository ppa:webupd8team/java
you@you:~$ sudo apt-get update
[更新 apt-get 的下载列表，需要输入你的计算机密码]
you@you:~$ sudo apt-get install oracle-java8-installer
[安装 Java 1.8，下面是验证安装是否成功的命令]
you@you:~$ java -version
java version "1.8.0_144"
Java(TM) SE Runtime Environment (build 1.7.0_67-b01)
Java HotSpot(TM) 64-Bit Server VM (build 24.65-b04, mixed mode)
```

（2）安装 Scala 的命令如程序清单 11-8 所示。

程序清单 11-8 在 Ubuntu 上安装 Scala

```
you@you:~$ sudo wget www.scala-lang.org/files/archive/scala-2.11.11.deb
[下载 Scala 2.11，需要输入你的电脑密码]
you@you:~$ sudo dpkg -i scala-2.11.8.deb
[安装 Scala 2.11，下面是验证安装是否成功的命令]
you@you:~$ scala -version
Scala code runner version 2.11.11 -- Copyright 2002-2017, LAMP/EPFL
```

（3）从 Spark 官方网站下载已经编译好的压缩包并解压就可以完成 Spark 的安装。它的启动命令如程序清单 11-9 所示。

程序清单 11-9 在 Ubuntu 上安装 Spark

```
[首先进入 Spark 的安装路径，也就是解压缩之后文件所在路径]
you@you:~$ ./bin/pyspark --master local[*]
[在本地启动 Spark，可以得到如下的画面]
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.1.0
```

书籍是人类进步的阶梯

```
 9 |       /_/
10 |
11 | Using Python version 2.7.9 (default, Feb 22 2017 21:10:11)
12 | SparkSession available as 'spark'.
```

11.1.2 从 MapReduce 到 Spark

从纯粹的技术角度来讲，大数据涉及两个方面：分布式存储系统和分布式计算框架。前者的理论基础是 Google 文件系统[5]（Google File System，GFS 或 GoogleFS），而后者的理论基础是 MapReduce。下面着重讨论计算框架 MapReduce 以及它和 Spark 之间的继承发展关系。

MapReduce 框架正如其名，它由两个步骤组成：一个是 Map，另一个是 Reduce。为了更直观地理解这两个步骤的作用，我们先来看一个简单的 word count 的例子。如图 11-4 所示，字符集存储在分布式的文件系统里，其中机器 1 存储了一个字符“a”；机器 2 存储了两个字符“a”和“b”；机器 3 存储了一个字符“b”。现在想要计算每个字符出现了多少次，如果所有的数据都存储在一台机器上，那么这就是一个非常简单的问题。但现在数据分散在不同的机器上，每台机器直接独立运算，只能得到自身数据的计算结果，因此需要按如下的步骤将集群上的机器整合起来。

（1）在每台机器上，各自独立地将需要统计的字符转换为所谓的 key-value 形式（key-value pair）。比如在机器 1 上将字符“a”转换为（“a”，1），其中“a”为 key，而 1 为 value。这里的 1 表示字符“a”出现了 1 次。这一步操作就是 Map。

（2）对于第 1 步中生成的 key-value 对，根据它的 key 值，将数据发送到相应的机器上（具体发送到哪台机器由 MapReduce 的内部算法决定），比如 key 等于“a”的数据都发送到机器 1、key 等于“b”的数据都发送到机器 3。这个数据移动的过程被称为 shuffle。

（3）数据经过 shuffle 之后，相同字符肯定都在同一台机器上，那么就可以在此基础上做字符计数运算了。事实上按 key 的值分组，将 value 值加起来就等于字符出现的次数。这一步操作就是 Reduce[6]。

---

[5] 对于分布式存储系统，大众更为熟知的是 HDFS（Hadoop Distributed File System）。它是开源项目 Hadoop 的一部分，而它的理论基础正是 GFS。

[6] 在 MapReduce 框架下，数据的格式都是 key-value 形式，其中 key 有两个作用：一方面它被用作统计的维度，类似于 SQL 语句里面的 group by 字段，比如正文例子里的字符；另一方面它又被用作数据的“指南针”决定数据将被发送到哪台机器上。而后者是分布式计算框架的核心。

在某些计算场景下，计算本身不需要 key 值，或者说不需要 Map 这一步，比如对一个数字数组求和。这种情况下，系统会自动地生成一个 Map 用于将数据转换为 key-value 形式，这时 key 值就只被用作数据的“指南针”。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

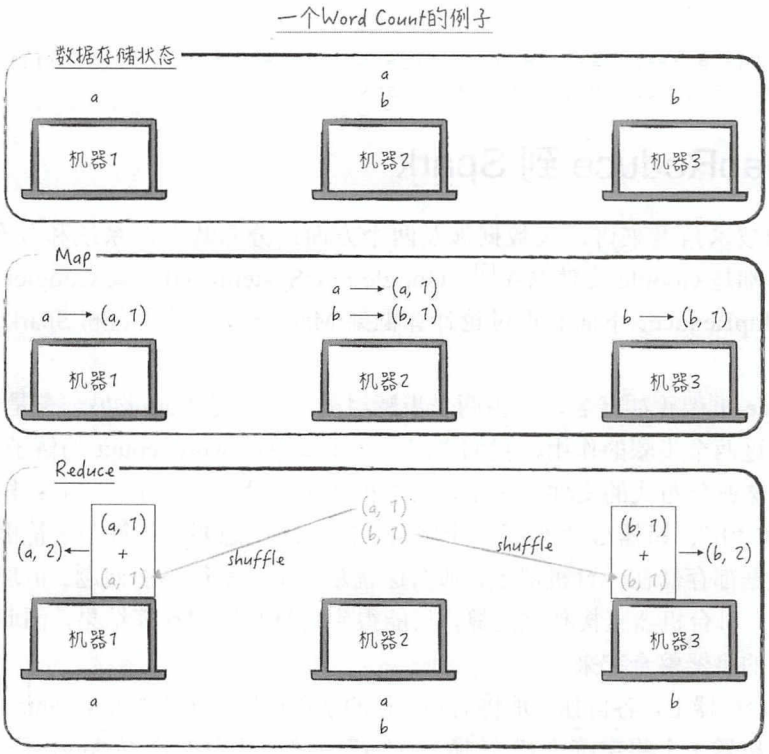


图 11-4

总结一下，MapReduce 里的 Map 步骤是在不同机器上独立且同步运行的，它的主要目的是将数据转换为 key-value 的形式；而 Reduce 步骤是做聚合运算，它也是在不同机器上独立且同步运行的。Map 和 Reduce 中间夹杂着一步数据移动，也就是 shuffle，这步操作会涉及数量巨大的网络传输（network I/O），需要耗费大量的时间。

由于MapReduce的框架限制，一个MapReduce任务只能包含一次Map和一次Reduce[7]，计算完成之后，MapReduce 会将运算结果写回到磁盘中（更准确地说是分布式存储系统）供下次计算使用。如果所做的运算涉及大量循环，比如估计模型参数的梯度下降或随机梯度下降算法就需要多次循环使用训练数据（算法的具体细节请参考 6.2 节和 6.4 节），那么整个计算过程会不断重复地往磁盘里读写中间结果。这样的读写数据会引起大量的网络传输以及磁盘读写，极其耗时，而且它们都是没什么实际价值的废操作。因为上一次循环的结果会立马被下一次使用，完全没必要将其写入磁盘。

[7] MapReduce 框架其实包含 5 个步骤：Map、Sort、Combine、Shuffle 以及 Reduce。这 5 个步骤中最重要的就是 Map 和 Reduce，这也是和 Spark 最相关的两步，因此这里只讨论这两个步骤。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

有实验结果显示，用 MapReduce 框架训练模型的性能极差，因为有 90%以上的时间都用于数据的读写（I/O），如图 11-5 所示。

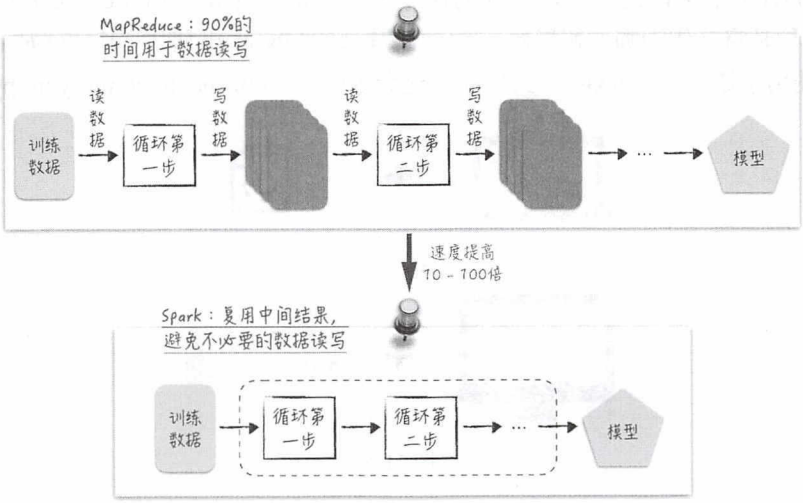


图 11-5

从上面的分析可以得到，整个算法的瓶颈是不必要的数据读写，而 Spark 主要改进的就是这一点[8]。具体地，Spark 延续了 MapReduce 的设计思路：对数据的计算也分为 Map 和 Reduce 两类。但不同的是，一个 Spark 任务并不止包含一个 Map 和一个 Reduce，而是由一系列的 Map、Reduce 构成。这样，计算的中间结果可以高效地转给下一个计算步骤，提高算法性能。虽然 Spark 的改进看似很小，但实验结果显示，它的算法性能相比 MapReduce 提高了 10～100 倍。

11.1.3 运行 Spark

介绍完 Spark 的理论基础之后，让我们再次回到编程实践上来，讨论 Spark 的运行方法。正如前面介绍的，Spark 提供包括多种语言的 API，不同语言的 API 有一些差别，但整体来说，功能是差不多的。限于篇幅，本书只讨论 Spark 的 Python API（PySpark）。

PySpark 与 Python 非常类似，有两种使用方法，如图 11-6 所示。

- 一种是利用它所提供的实时交互命令窗口（PySpark shell）。需要注意的是，PySpark

[8] Spark 的创新还有很多，比如在训练模型时，它会将数据保持在内存里。因为训练模型会重复使用训练数据，所以这样会有效地提高算法性能，这也是 Spark 最出名的所谓“内存计算”（in memory computing）。但就提升算法性能而言，作用最大的还是正文中讨论的运算框架上的改进。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

shell 其实就是之前章节中一直讨论的 Python shell，只不过它自动地引入了第三方库 Spark，因此可以在 PySpark shell 里执行任意 Python 语句以及使用 Spark 提供的函数。PySpark 的启动方式已在 11.1.1 节中讨论过了，具体的命令为“./bin/pyspark --master local[*]”。

- 另一种是将其作为脚本解释器，运行已编辑好的 PySpark 脚本（其实就是 Python 脚本，只不过其中使用了第三方库 Spark）。具体的提交命令如下“./bin/spark-submit [your pyspark script]”。

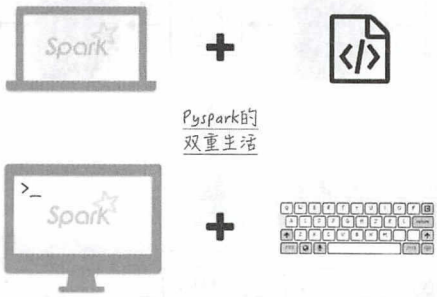


图 11-6

PySpark 有两种运行方式，一种是本地运行模式（local mode），一种是集群运行模式（cluster mode）。这两种方式由参数“--master”控制。11.1.1 节中讨论的 PySpark 启动命令“./bin/pyspark --master local[*]”就表示在本地运行 PySpark；在集群上运行 PySpark 的命令取决于具体的集群环境，比如在 Hadoop 集群上（准确地说是在 Yarn[9]下面）的启动命令为“./bin/pyspark --master yarn”。

Spark 在这两种运行方式下的表现是几乎一样的[10]，因此我们可以先在本地运行并验证编写好的 PySpark 程序，然后不用做任何修改就能将其部署到分布式集群上运行，这样能极大地提高程序开发和测试的效率。

11.1.4 Spark DataFrame

本节将讨论 Spark 的核心组件之一——Spark DataFrame[11]。借助它，我们可以很轻松地

[9] Yarn（Yet Another Resource Negotiator）是 Hadoop 2.0 新引入的资源管理系统，它负责管理和分配分布式计算集群上的各种资源。PySpark 作为一个应用程序，在启动的时候，需要向 Yarn 申请所需的计算资源。

[10] Spark 在本地模式和集群模式的代码是完全一样的，但两者的表现还是有少许不同，主要体现在 shuffle 这一步，也就是说两者的数据传输机制不一样。在绝大部分情况下，这些差别并不影响程序的结果，但当使用 repartition 函数时就需要特别注意了，因为该函数在两种运行模式下的结果并不一样。

[11] 在之前章节使用第三方库 pandas 时，我们也接触过 DataFrame。事实上 Spark DataFrame 就是参考 pandas DataFrame 而设计的，因此两者的用法很相似，只是 pandas DataFrame 只能处理本地的“小”数据，而 Spark DataFrame 可以处理分布式环境下的大数据。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

完成对大数据的处理。事实上，Spark 编程的基础是 RDD（Resilient Distributed Datasets），它是一个程序上的抽象，代表着分布式的数据集。Spark 在它上面实现了大数据处理的核心算法，比如 Map 和 Reduce。但由于 RDD 提供的函数（API）不够简单和直接，需要有比较好的编程功底，因此 Spark 又在 RDD 上面封装了一层，变成 Spark DataFrame 方便用户使用。

1. Spark DataFrame 的读写

（1）Spark 中的 SparkSession 类是运行 Spark DataFrame 的基础[12]。启动 PySpark shell 时，系统已经自动创建了一个 SparkSession 的实例“spark”，通过它就可以定义相应的 Spark DataFrame 了。

（2）创建 DataFrame 的方法主要有 3 种，分别是：读取特定格式的数据，比如 json 或 parquet 数据，如程序清单 11-10 中第 3、4 行代码所示；读取数据库里的数据，比如读取 HIVE 里的某一张表，如第 6 行代码所示；将本地 Python 内的数据转换为 DataFrame，如第 8～10 行代码所示。

（3）DataFrame 的保存方法主要有两种：一是将数据保存为特定格式的数据，如第 12、13 行代码所示；二是将数据保存到外部数据库，比如 HIVE，如第 15 行代码所示。

程序清单 11-10 Spark DataFrame 的读写

```
1  | # PySpark shell 在启动时会自动创建变量 spark
2  | # 读取特定格式的数据
3  | >>> jsonDF = spark.read.json(...)
4  | >>> parquetDF = spark.read.parquet (...)
5  | # 读取 HIVE 表
6  | >>> hiveDF = spark.sql("select * from table")
7  | # 将本地 Python 数据转换为 DataFrame，比如将 pandas DataFrame 转换为 Spark DataFrame
8  | >>> import pandas as pd
9  | >>> data = pd.DataFrame([{"id": "0", "name": "tang"}, {"id": "1", "name": "an"}])
10 | >>> df = spark.createDataFrame(data)
11 | # 将 DataFrame 保存为特定格式的数据
12 | >>> jsonDF.write.json(...)
13 | >>> parquetDF.write.parquet(...)
14 | # 将 DataFrame 保存到 HIVE 表
15 | >>> hiveDF.write.saveAsTable(...)
```

2. Spark DataFrame 的操作

如果读者对 Python 中的 pandas DataFrame 比较熟悉，会发现 Spark DataFrame 与它非常相似。

（1）选择数据中一列或多列。这个操作的具体实现有两种，一种是使用类似 SQL 的 API：select，如程序清单 11-11 中第 14～20 行代码所示。另一种是使用类似 pandas DataFrame 的 API，如第 21、22 行代码所示。值得注意的是，使用第一种方法返回的是 Spark DataFrame，

[12] 从底层实现上来说，Spark 程序的基础和入口是 SparkContext 类，它对应的数据抽象是 RDD。为了方便用户使用，Spark 在 SparkContext 上面封装了一层，变成了正文中的 SparkSession，而后者对应的数据抽象是 Spark DataFrame。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

而第二种方法返回的是 Column。

（2）在已有 DataFrame 的基础上增加一列。通过 withColumn 函数可以完成这个操作，具体的实现如第 24～37 行代码所示。

程序清单 11-11 Spark DataFrame 的操作（1）

```
1  | >>> import pandas as pd
2  | >>> from pyspark.sql import functions as F
3  | >>> data = pd.DataFrame([{"id": "0", "name": "tang"}, {"id": "1", "name":
"an"}])
4  | >>> df = spark.createDataFrame(data)
5  | # 展示 DataFrame 中最初的 20 行
6  | >>> df.show()
7  | +---+----+
8  | | id|name|
9  | +---+----+
10 | |  0|tang|
11 | |  1|  an|
12 | +---+----+
13 | # 选择数据中的 ID 列
14 | >>> df.select("id").show()
15 | +---+
16 | | id|
17 | +---+
18 | |  0|
19 | |  1|
20 | +---+
21 | >>> df["id"] ## 或者 df.id
22 | Column<id>
23 | # 增加一列
24 | >>> df.withColumn("new_id", 2 * df.id + 1).show()
25 | +---+----+------+
26 | | id|name|new_id|
27 | +---+----+------+
28 | |  0|tang|   1.0|
29 | |  1|  an|   3.0|
30 | +---+----+------+
31 | >>> df.withColumn("const", F.lit(10)).show()
32 | +---+----+-----+
33 | | id|name|const|
34 | +---+----+-----+
35 | |  0|tang|   10|
36 | |  1|  an|   10|
37 | +---+----+-----+
```

（3）对 DataFrame 里的数据做筛选。这个操作可由函数 filter 来完成，如程序清单 11-12 中第 2～7 行代码所示。

（4）对数据做聚合运算。Spark DataFrame 提供相当丰富的类似 SQL 的聚合运算 API，比如 min、max，如第 9～14 行代码所示。借助这些 API 可以很方便地完成对数据的统计以及其他比较复杂的聚合运算。

书籍是人类进步的阶梯

程序清单 11-12 Spark DataFrame 的操作（2）

```
# 根据 ID 字段筛选数据
>>> df.filter(df.id == "0").show()
+---+----+
| id|name|
+---+----+
|  0|tang|
+---+----+
# 对数据做聚合运算
>>> df.groupBy().agg(F.max("id"), F.min("id")).show()
+-------+-------+
|max(id)|min(id)|
+-------+-------+
|      1|      0|
+-------+-------+
```

11.1.5 Spark 的运行架构

本节我们将离开模型这个话题，讨论 Spark 的运行架构。虽然这部分内容与模型没有直接关系，但对深入理解 Spark，特别是 PySpark 很有帮助。

Spark 是分布式计算引擎，在实际生产环境中，一个 Spark 程序通常运行在由多台机器组成的计算机集群上（需要注意的是，这里所说的机器并不指物理意义的计算机，而是虚拟机。比如在本地运行 Spark 时，系统会在本地创建多个虚拟机供 Spark 使用）。如图 11-7 所示，为了协调这么多台机器同时工作，Spark 将机器分为两类：driver 和 executor。

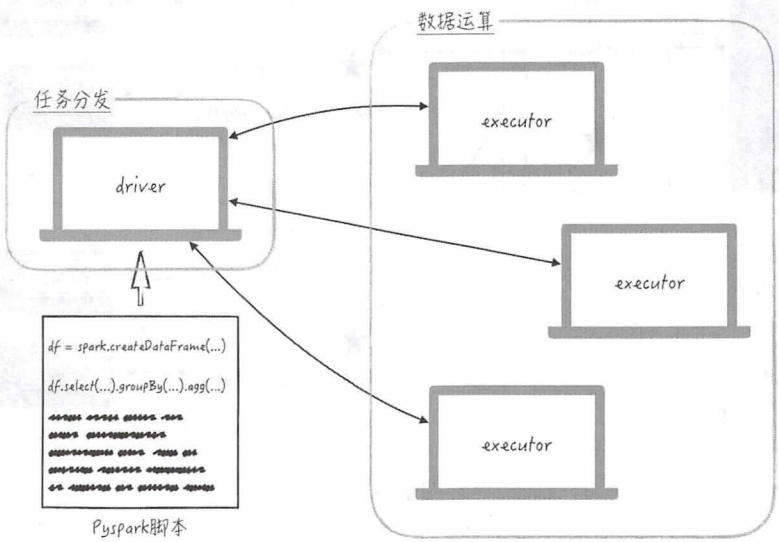


图 11-7

书籍是人类进步的阶梯

- 一个 Spark 程序只有一个 driver，就是提交 PySpark 脚本或者启动 PySpark shell 的机器。它的作用是控制并协调集群里的 executor：向后者分发任务，并监督每个任务的运行情况，因此 driver 并不进行任何实质上的数据计算。对软件开发比较熟悉的读者可以将 driver 形象地理解为项目经理。
- 集群里除了 driver，其他机器都是 executor，一个 Spark 程序通常有多台 executor。它们的作用是接收 driver 的命令进行运算，并将运算结果反馈给 driver。

正如本节一开头提到的，Spark 的核心功能是用 Scala 语言实现的，那么 PySpark 是如何与 Scala（JVM）进行交互的呢？用 Python 代码定义的计算步骤是在 Python 中进行的吗？为了回答这两个问题，让我们来看看 PySpark 的运行架构，如图 11-8 所示。

- 从宏观的角度来讲，PySpark 也将集群中的机器分为 driver 和 executor。不过 driver 和 executor 里不仅有 JVM 进程（Scala 程序），还有 Python 进程。
- 当一个 PySpark 脚本被提交给 driver 后，driver 上的 Python 进程会将代码中需要分布执行的部分（通过 Spark DataFrame API 传入的代码，比如图 11-8 中通过 map 接口传入的“f(x)”）经过编码后传给 driver 上的 JVM 进程，如图 11-8 中的五角星所示。
- 而 driver 上面的 JVM 进程会根据上面讨论的 Spark 框架，将需要分布执行的任务分发给各台 executor，并监控它们的运行情况。

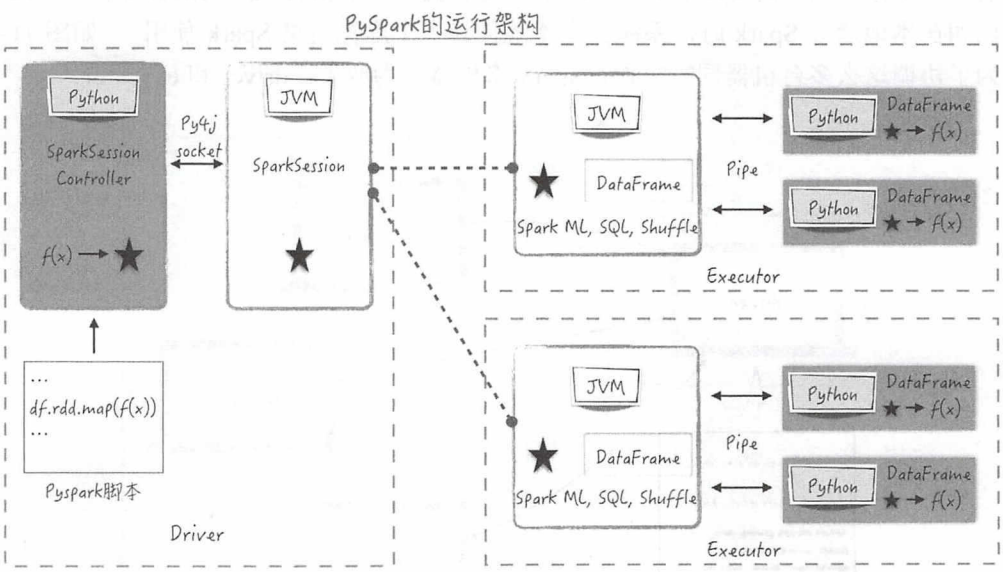


图 11-8[13]

[13] 图片参考自 Spark summit 2015。

书籍是人类进步的阶梯

- executor 拿到由 driver 分发的任务后，会根据任务的具体内容决定在哪里执行相应的运算：如果所接收的任务是有关 Spark ML 的，比如调用 Spark 的机器学习算法库，则直接在 JVM 里面进行数据计算，类似地还有 Spark SQL 和 shuffle 等操作；如果是其他类型的运算，比如通过 map 接口传入的函数“f(x)”，则 executor 里的 JVM 进程将把接收到的任务传给 Python 进程，由后者去执行具体的计算。

通过上面这个比较精妙的设计，PySpark 既最大限度地沿用了 Spark 本身的核心设计，也实现了 Python 代码的分布式执行。而后者保证了 PySpark 能很好地和 Python 已有的算法库进行交互，比如在分布式的环境下训练 scikit-learn 里的模型，这部分内容的具体细节请参考 11.3.2 节。

11.2 最优化问题的分布式解法

在介绍完分布式计算工具 Apache Spark 之后，让我们重新回到分布式机器学习上来，看看应该如何“分布式”地估算模型参数。

11.2.1 分布式机器学习的原理[14]

第 6 章曾讨论了，在一台计算机上如何利用梯度下降法（gradient descent）估算模型参数，这里先简单回顾一下这个算法。

对于绝大多数的机器学习模型，它的损失函数都能写成和的形式，表示模型的整体损失等于每点损失之和。因此不妨假设相应的损失函数为$L = \sum_i l(w, \boldsymbol{X}_i, y_i)$，其中，$l(w, \boldsymbol{X}_i, y_i)$表示模型在第$i$个数据点的损失，$w$为模型参数，$\boldsymbol{X}_i$为自变量，$y_i$为被预测量。根据梯度下降法，模型参数$w$的迭代公式为：

$$w_{k+1} = w_k - \eta \sum_i \frac{\partial l}{\partial w}(w_k, \boldsymbol{X}_i, y_i) \tag{11-1}$$

在公式（11-1）中，参数η为学习速率（learning rate），表示每次更新参数的步长。数学上可以证明，根据上面公式的不断迭代，则$\{w_k\}$会收敛于损失函数的某个极小值点。仔细分析这个算法会发现，算法本身并没有要求数据必须存放在一台机器上，也没有要求相关的计算必须在一台机器上完成，因此这个算法完全可以应用于分布式的环境。

如图 11-9 所示，可以将梯度下降法用分布式计算框架 MapReduce 来实现，在每次迭代中：

- Map 这一步将第i个数据$(\boldsymbol{X}_i, y_i)$转换为单点损失梯度$\frac{\partial l}{\partial w}(w_k, \boldsymbol{X}_i, y_i)$；

[14] 限于篇幅，这里只讨论监督式学习的分布式实现。对于非监督式学习，计算原理和工程实现都是类似的。

书籍是人类进步的阶梯

- Reduce 这一步将 Map 的结果进行加和。

MapReduce 的运算完成之后，用得到的结果去更新参数估计值，并进入迭代的下一步。

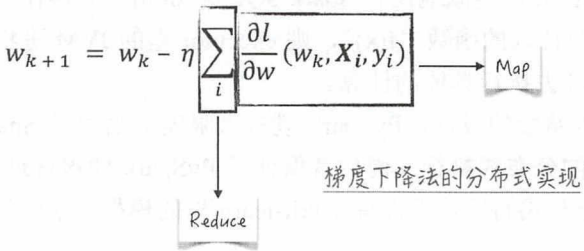


图 11-9

在大数据下使用梯度下降法训练模型会发现这个算法的效率比较低下，因为在梯度下降法中，每一次迭代都需要计算所有的训练数据，非常耗时。正因为如此，在实际生产中，我们常使用原理类似但效率更高的随机梯度下降法（stochastic gradient descent）。这个算法的迭代公式与梯度下降法几乎一模一样，唯一的区别在于，每次迭代并不使用所有数据来计算损失函数的梯度，而是用部分数据的单点损失梯度来近似损失函数的梯度。具体如公式（11-2）所示，其中，m小于训练数据的个数。

$$w_{k+1} = w_k - \eta \sum_{j=1}^{m} \frac{\partial l}{\partial w}(w_k, \boldsymbol{X}_j, y_j) \tag{11-2}$$

与图 11-9 所示的算法类似，随机梯度下降法也可以用 MapReduce 框架来实现，而且具体的算法步骤也与梯度下降法类似。

11.2.2 一个简单的例子

本节以线性回归模型为例，来看看如何利用 Spark 实现分布式的梯度下降法。值得注意的是，虽然实现的算法是分布式的，但代码也能在 Spark 的单机模式下运行。

对于线性回归模型，它的损失函数如公式（11-3）所示（具体细节请参考第 4 章）。

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{X}_i\boldsymbol{\beta})^2 = \frac{1}{n}\sum_{i=1}^{n} l(\boldsymbol{X}_i, y_i) \tag{11-3}$$

在上面的公式中，$\boldsymbol{\beta}$为模型参数，对它求偏导数，可以得到损失函数的单点梯度为：

$$\frac{\partial l}{\partial \beta}(\boldsymbol{X}_i, y_i) = -2(y_i - \boldsymbol{X}_i\boldsymbol{\beta})\boldsymbol{X}_i \tag{11-4}$$

根据公式（11-4）并结合 11.2.1 节中的分析，在分布式环境下训练线性回归模型就比较容易实现了，如程序清单 11-13 所示。

（1）Spark 程序的入口是 SparkSession 类，因此在开始计算之前，需要创建一个 SparkSession 的实例，如第 8 行代码所示（如果使用 PySpark shell，系统已经将这一步自动完成了）。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

（2）跟单机版的梯度下降法一样，首先随机生成模型参数的初始值，如第 27 行代码所示。

（3）为了方便后面的计算，我们将被预测量命名为“y”，并将所有的自变量放在一起，组成一个 np.array，命名为“x”，如第 29、30 行代码所示。

（4）由于梯度下降法会多次重复使用数据，因此将训练数据存放在内存里[15]，这样可以节省每次从硬盘里读入数据的时间，如第 32 行代码所示。这是 Spark 的主要创新点，也是它被称为内存计算的原因。

（5）在 MapReduce 框架中，Map 这一步将计算损失函数的单点梯度（公式（11-4）），它的实现如第 36 行代码所示；而在 Reduce 这一步将计算损失函数单点梯度的和，如第 38 行代码所示。需要注意的是，第 37 行代码将单点梯度“x”转换为“(x, 1)”，这里的 1 用于计算数据的个数，而后者将用于计算平均单点梯度，如第 40 行代码所示。

（6）最后是根据计算得到的梯度，更新模型参数，如第 42 行代码所示。

程序清单 11-13 分布式线性回归模型[16]

```
1  import numpy as np
2  from pyspark.sql import SparkSession
3
4  def startSpark():
5      """
6      创建 SparkSession，这是 Spark 程序的入口
7      """
8      spark = SparkSession.builder.appName("gd_example").getOrCreate()
9      return spark
10
11 def gradientDescent(data, labelCol, featuresCol, stepSize=1, maxIter=100):
12     """
13     利用梯度下降法训练线性回归模型
14
15     参数
16     ----
17     data : Spark.DataFrame，训练模型的数据
18
19     labelCol : str，被预测量的名字
20
21     featuresCol : list[str]，自变量的名字列表
22
23     stepSize : float，梯度下降法里的学习速率
24
25     maxIter : float，梯度下降法的迭代次数
26     """
```

[15] 这样的说法并不太严谨，因为计算机在做任何运算时，都会将数据放入内存，但默认情况下，一次运算完成之后，计算机会将内存里的数据清空，以便下次计算时使用。在 Spark 中，将数据放入内存（cache 命令）指的是在运算完成之后，强行让留在内存里的数据不清空。

[16] 完整的实现请参考随书配套的代码/ch11-spark/gradient_descent_example.py。

仅供非商业用途或交流学习使用

```
    w = np.random.randn(len(featuresCol))
    # 数据转换为dict形式，并将自变量转换为一个np.array
    df = data.rdd.map(
        lambda item: {"y": item[labelCol], "x": np.array([item[i] for i in
        featuresCol])})
    # 将数据放到内存里，提高运算速度
    df.cache()
    for i in range(maxIter):
        # 计算线性回归模型的单点梯度以及数据的个数
        _gradient, num = df\
            .map(lambda item: -1 * (item["y"] - w.dot(item["x"])) * item["x"])\
            .map(lambda x: (x, 1))\
            .reduce(lambda a, b: (a[0] + b[0], a[1] + b[1]))
        # 计算损失函数的平均单点梯度
        gradient = _gradient / num
        # 更新模型参数
        w -= stepSize * gradient
    return w
```

Spark ML 其实已经实现了分布式的最优化算法，比如随机梯度下降法等，可以直接调用这些算法来训练模型（唯一需要做的就是定义损失函数以及损失函数的单点梯度）。但由于 Spark 的核心功能是由 Scala 实现的，因此这些最优化算法只有 Scala 版本。有兴趣的读者可参考 GitHub 上的开源代码（具体路径为 src/main/scala/org/apache/spark/mllib/optimization）。

11.3 大数据模型的两个维度

在大数据环境下，常常遇到两种不同维度的分布式机器学习问题。一种是训练模型的数据集很大，单台计算机根本无法处理，只能依靠集群的力量，分布式地训练模型，这种情况被称为数据量维度。另一种是需要训练的模型个数很多，但每个模型都比较“小”，单台计算就能处理，这种情况被称为模型数量维度。对于这两种不同的应用场景，Spark 都能很好地处理。下面将讨论具体的细节。

11.3.1 数据量维度

能在大数据量下训练机器学习模型是 Spark 的最大亮点，11.2 节中讨论的最优化问题的分布式解法也是针对数据量这个维度而言的。截止到 2.1.0 版本，Spark 针对这类问题的开源算法库有两个，分别是 Spark MLlib（MLlib RDD-based API）和 Spark ML（MLlib DataFrame-based API）[17]，两者的差别如下。

[17] 如果想要在 PySpark 里使用这两个组件，需要确保集群上每台机器安装好了第三方库 NumPy（1.4 版本或者更新）。

仅供非商业用途或交流学习使用

Spark MLlib 是较早版本的分布式机器学习算法库，它所用的训练数据是用 RDD 抽象的，目前已经停止更新了，而且 Spark 开发社区预计将在 3.0 版本中删除这部分内容。

Spark ML 是较新版本的算法库，它所用的训练数据是用 Spark DataFrame 抽象的。Spark ML 里的很多底层部件都复用了 Spark MLlib 的实现，比如最优化问题算法。

可以这样理解，Spark ML 是 Spark MLlib 的升级版本，但由于还在升级的过程中，因此保留了较早版本的算法库。从实用性上来讲，Spark MLlib 不太适合初学者使用，因为它要求使用者有较好的模型理解能力以及编程技巧（可参考后面 11.4 节里的例子），当然它的灵活性更强；而 Spark ML 更加用户友好，更易使用，如图 11-10 所示。Spark ML 是 Spark 开源社区今后重点开发的模块，将不断有新的算法加入，因此这也是本节讨论的重点。

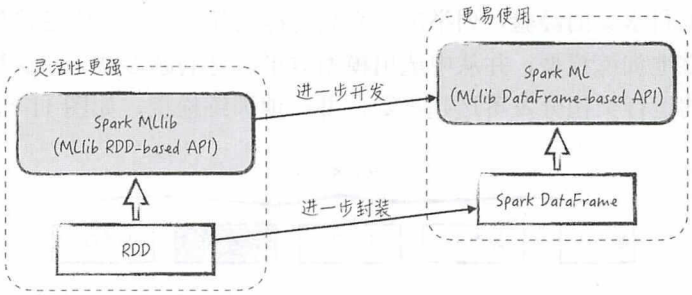


图 11-10

在设计上，Spark ML 参考了开源算法库 scikit-learn 的架构，因此从算法 API 的角度来讲，这两者的差异并不大，使用过 scikit-learn 的读者会对 Spark ML 感到很熟悉，比如 Spark 官网上有关逻辑回归的例子。

程序清单 11-14 Spark 逻辑回归

```
from pyspark.ml.classification import LogisticRegression

# 读取数据
data = spark.read.format("libsvm").load("data/mllib/sample_libsvm_data.txt")
# 训练模型
lr = LogisticRegression()
lrModel = lr.fit(data)
# 输出模型结果
print("Coefficients: " + str(lrModel.coefficients))
print("Intercept: " + str(lrModel.intercept))
```

在模型的选择上，除了最经典的机器学习算法，比如逻辑回归、支持向量学习机等，Spark ML 更加注重在分布式环境下可行性和实用性都较强的模型，比如用于大规模推荐系统的 ALS 算法（这个模型的细节请参考 10.5.4 节）、用于自然语言处理的 Word2Vec 算法、用于文本分类的 LDA（Latent Dirichlet Allocation）模型等。限于篇幅，这里就不讨论这些模型的

仅供非商业用途或交流学习使用

细节了，有兴趣的读者请参考 Spark 的官方网站。

11.3.2 模型数量维度

在大数据的实际生产中，常会遇到需要一次性训练大量模型[18]的情况。其中最常见的场景是模型超参数调试（hyperparameter tuning），具体细节如下。

在搭建模型时，我们会碰到一些超参数（hyperparameter），比如第 4.3.3 节中讨论的惩罚项权重，这是模型本身的超参数，又比如随机梯度下降法里涉及学习速率、迭代次数等（细节请参考第 6 章或 11.2.2 节），这些是工程实现上的超参数。超参数与普通的模型参数有所不同，不能使用最优化算法求解它们。因此对于超参数，通常使用网点搜寻（grid search）的方法：根据数据科学家的经验，列举出一系列可能的超参数，组成备选集；然后遍历这个备选集，各自独立地训练模型，并从中选出模型效果最好的超参数。在这种应用场景下，可以利用 PySpark 的运行架构分发备选超参数，并行地训练模型，如图 11-11 所示。

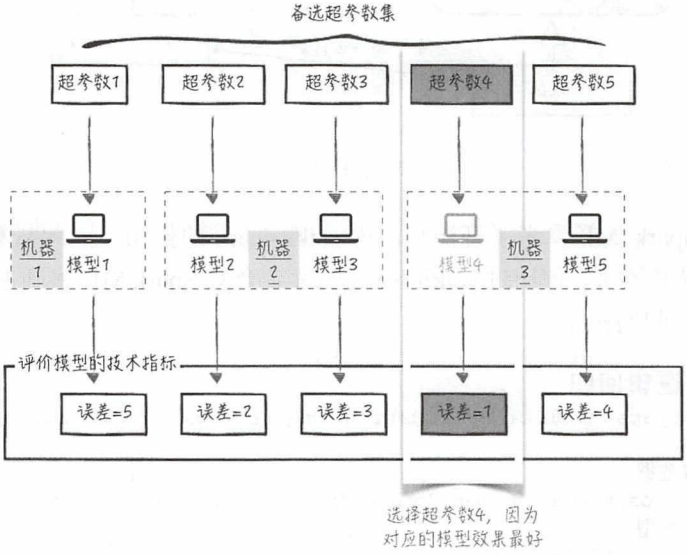


图 11-11

针对不同的单机算法库，有专门的开源项目实现了上述的并行训练模型的功能，比如针对 scikit-learn 的 spark-sklearn 以及针对 TensorFlow 的 TensorFlowOnSpark[19]。这两个开源项

[18] 从可行性的角度考虑，我们要求这些模型都是比较“小”的模型，单台机器就能完成模型训练。如果这些模型本身就是很“大”，需要分布式的集群才能训练，那么无论是从理论上还是从实践上，都没有特别高效的解决方案。

[19] 这两个开源项目都能在开源社区 GitHub 上找到，相应的地址为 databricks/spark-learn 和 yahoo/TensorFlowOnSpark。

仅供非商业用途或交流学习使用

目虽然针对的单机算法库不同，但分发任务的核心实现却是一样的，限于篇幅，这里仅以 spark-sklearn 为例，说明相关的实现细节。

- 首先通过 SparkContext 的 parallelize[20]函数将本地的备选超参数转换为 RDD。
- 然后利用 Spark 的分布框架将备选超参数分发到不同的 executor 上，并调用 sklearn 的算法，根据分配到的超参数训练模型。
- 最后根据模型效果，选择最优的超参数，具体的过程如图 11-12 所示，其中，Param1、Param2 和 Param3 表示备选的超参数。

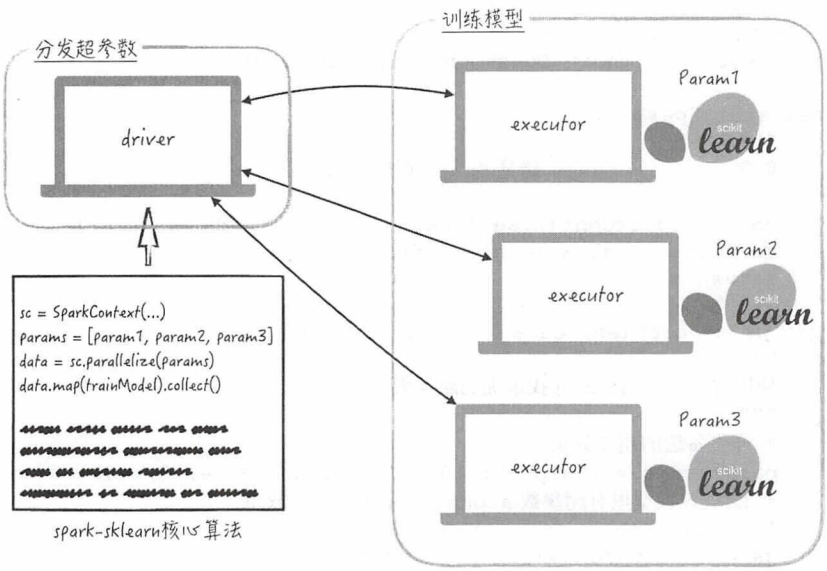


图 11-12

下面以 Lasso 回归（带有 L1 惩罚项的线性回归模型，具体的细节请参考 4.3.3 节）为例，展示如何使用 spark-sklearn。

先简单介绍一下模型的背景，假设数据里有 3 个变量，分别为x, y, z。其中，x, z为自变量，y为因变量，它们之间的关系为$y = x + \varepsilon$，ε为随机扰动项。也就是说，变量z是不相关变量，将它放入模型会损害模型的有效性。而 Lasso 模型可以通过惩罚项来解决这个问题，但需要通过网点搜寻来找到最佳的惩罚项权重。具体的实现如程序清单 11-15 所示。

（1）为了使用 spark-sklearn，首先需要创建 SparkContext 的实例，后者是整个 Spark 程序的入口，如第 9、10 行代码所示。也可以如程序清单 11-13 中第 8 行所示创建 SparkSession

[20] 与 SparkSession 的 createDataFrame 函数类似，parallelize 函数把 driver 上的数据转换为 RDD。事实上，createDataFrame 函数是 parallelize 函数的进一步封装。

仅供非商业用途或交流学习使用

的实例“spark”，而“spark._sc”就是相应的 SparkContext 的实例。

（2）跟 scikit-learn 里的代码步骤一样，首先生成备选的超参数集，如第 18 行代码所示，然后将 SparkContext、Lasso 模型以及备选超参数传给 GridSearchCV，如第 21 行代码所示。需要注意的是，spark_sklearn 中的 GridSearchCV 除了多一个 SparkContext 参数外，其余部分与 scikit-learn 中的一模一样。

（3）创建好 GridSearchCV 对象后，通过 fit 函数来并行地训练模型，如第 22 行代码所示。

程序清单 11-15 并行训练模型[21]

```
 1 | from sklearn.linear_model import Lasso
 2 | from spark_sklearn import GridSearchCV
 3 | from pyspark import SparkContext, SparkConf
 4 |
 5 | def startSpark():
 6 |     """
 7 |     创建 SparkContext，这是 Spark 程序的入口
 8 |     """
 9 |     conf = SparkConf().setAppName("grid search example")
10 |     sc = SparkContext(conf=conf)
11 |     return sc
12 |
13 | def gridSearch(sc, data, label, features):
14 |     """
15 |     使用 grid search 寻找最优的超参数
16 |     """
17 |     # 产生备选的超参数集
18 |     parameters = {"alpha": 10 ** np.linspace(-4, 0, 45)}
19 |     # Lasso 模型里有超参数 alpha，表示惩罚项的权重
20 |     la = Lasso()
21 |     gs = GridSearchCV(sc, la, parameters)
22 |     gs.fit(data[features], data[label])
23 |     return gs
```

运行上面的代码，可以得到最佳的超参数是“alpha=0.152”，这与 4.3.3 节中的结论是相符的。在最优超参数下，相应的模型为$y = 1.009x - 0.018$，这个结果很接近真实情况。

11.4 开源工具的另一面

到目前为止，本书讨论了很多的开源算法库（本书里的模型实现也大多基于这些开源工具），比如针对统计分析的 Statsmodels、针对机器学习的 scikit-learn，以及针对分布式环境的 Spark ML 和 Spark MLlib。在之后的两章，我们还将针对神经网络以及深度学习讨论另一

[21] 完整的实现请参考随书配套的代码/ch11-spark/hyperparameters_tuning.py。

仅供非商业用途或交流学习使用

个开源算法库——TensorFlow。

之所以花大篇幅讨论这些开源工具是因为经过近 10 年的快速发展，开源社区已经成为了数据科学领域并不可少的一部分。在实际生产中，数据科学家们常常借助这些算法库来完成数据分析和模型搭建的任务，就像本书一样。

虽然这些算法库都十分优秀，而且是免费开源的，但它们跟世间万物一样，并不是百分之百的完美。在实际应用中，常常会碰到各种各样的“坑”。为了更加形象地说明这个问题，下面先来看一个简单的例子。

11.4.1 一个简单的例子

第 4 章讨论了线性回归模型，它是一个非常简单和基础的模型。但即使是针对这么一个简单到“让人觉得没有什么实用价值”的模型，在使用开源工具时，也需要十分小心。

例如对于如图 11-13a 所示的数据，因变量y与自变量x之间有比较明显的线性关系。在不进行任何参数调优的情况下，如果使用 scikit-learn 中提供的线性回归[22]，则可以得到不错的模型结果；但如果使用 Spark MLlib（Spark 的版本为 2.1.0）中提供的同样模型，则效果较差。或者看一个更极端的例子，如图 11-13b 所示，由 Spark MLlib 训练出来的模型完全不对，预测结果甚至比随机猜测更糟糕。

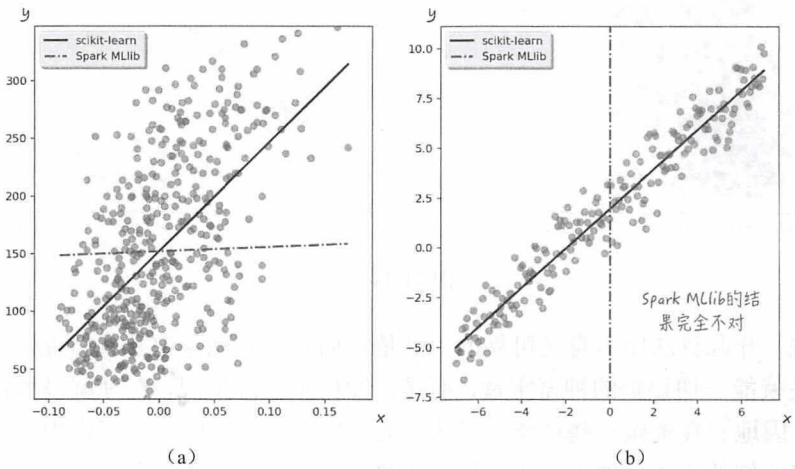


图 11-13[23]

[22] scikit-learn 中对应的模型为 sklearn.linear_model.LinearRegression，而 Spark MLlib 中对应的模型是 pyspark.mllib.regression.LinearRegressionWithSGD。

[23] 完整的实现请参考随书配套的代码/ch11-spark/sparkml_vs_sklearn.py。

仅供非商业用途或交流学习使用

这样的结果令人震惊，因为 Spark 在 2014 年就成为了 Apache 软件基金的顶级项目，在开源社区享有盛名，很多知名公司都为其背书。但就是这么有名的开源工具却连一个最简单的线性回归问题都解决不好[24]。那么开源算法库还可靠吗？我们还能在生产中使用这些算法库吗？

在回答这两个尖锐的问题之前，让我们来分析一下产生上面结果的原因（针对 Spark MLlib）。从技术层面上来讲，这个原因十分明显：在不进行参数调优的情况下，最优化算法（随机梯度下降法）没有收敛。对于图 11-13a，算法在到达收敛点之前就停止迭代了，而对于图 11-13b，算法向外发散了，根本没往收敛点靠近。

解决这个问题的方法有两种，一种是使用网点搜寻的方法，找出最优的工程实现超参数（细节请参考第 6 章）；另一种是将变量归一化，使算法收敛更加容易（细节请参考第 7.5.3 节）。使用这两种方法后，得到的结果如图 11-14 所示。

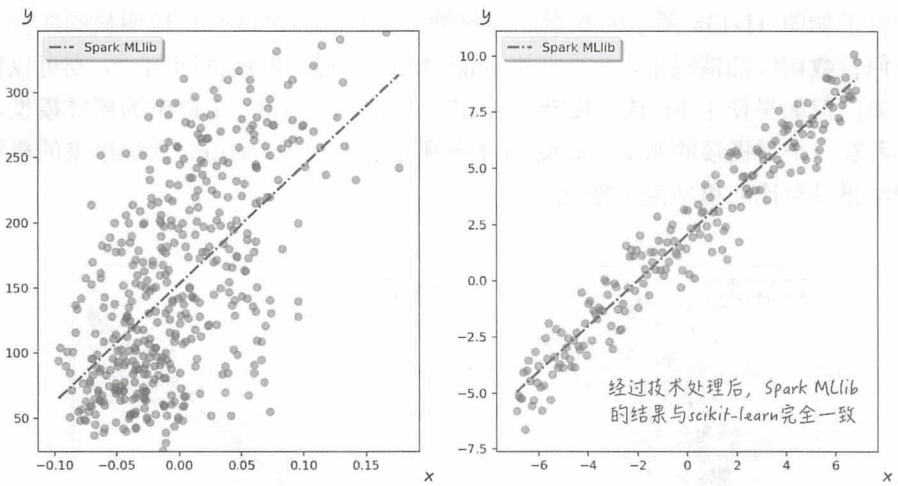


图 11-14[25]

也就是说，开源算法库本身是可靠的，只是它们并没有想象中那么智能。不能将它们当作按一下开关就能一切搞定的神奇宝盒，而是需要根据实际情况，在理解模型原理和工程实现的基础上，因地制宜地做一些调整。因此，虽然数据科学家们会在工作中大量使用第三方开源工具，但他们的主要技能点（或者最大优势）并不是会调用这些工具，而是真正地理解这些工具，比如它们工程实现的逻辑以及相应的优缺点，以便更好地借助它们搭建起预测效

[24] 虽然在 Spark 2.0.0 之后，Spark ML 里的线性回归模型解决了正文中的问题，但在相当长的时间内（2010~2016），Spark 提供的模型算法（包括但不限于线性回归）都是比较粗糙的。

[25] 完整的实现请参考随书配套的代码/ch11-spark/sparkml_vs_sklearn_solution.py。

果好的模型。当然这也是优秀数据科学家发挥作用、体现价值的地方。

11.4.2 开源工具的阿喀琉斯之踵

在图 11-13 的例子中，开源工具 scikit-learn 的表现比 Spark MLlib 要好很多。举这个例子不是为了证明 scikit-learn 比 Spark MLlib 好用，而是想告诉读者，不同开源工具的成熟度是不一样的，因此在解决问题是，应尽可能地选择成熟度更高的工具，比如针对“小”数据，scikit-learn 会更加可靠。

但即使是 scikit-learn 这个已经发展 10 年、被业界公认为最成熟的开源算法库，它其中也有很多十分严重的错误。比如 10.2.3 节中讨论的谱聚类（spectral clustering），截至本书编写时的 0.18 版本，scikit-learn 对这个模型的实现都是不准确的，这导致在过去 10 年里，使用它得到的结果都是错误且毫无意义的。

类似的例子还有很多，举这些例子的目的不在于批判 scikit-learn，而是用事实说明，即使再权威的开源工具都隐藏着不少致命的 bug[26]。如果盲目信任开源工具，可能会因为这些 bug 而导致非常严重的后果。而且从商业上来说，开源工具在法律上是免责的，也就是说，如果由于开源工具的错误，而导致了很严重的损失，是没有其他公司或个人来承担这个责任的。因此与传统意义上的商业软件不同，使用开源工具时，是使用者来为所用工具的准确性负责。在我看来这就是开源工具的阿喀琉斯之踵。

这些开源工具并不是使用者自己开发的，而且它们的架构通常较为复杂，不易理解。那么应该如何有效地检测这些工具呢？这个问题很复杂，而且据我了解，到目前为止并没有一个通用和成体系的解决方法。在实际中比较常用而且有效的方法是单元测试。由于模型涉及的数学知识比较深奥，需要人为设计模型场景和测试结果去验证代码逻辑是否正确，具体的细节请参考 9.4.5 节。

11.5 本章小结

本章讨论的重点是大数据的核心内容——分布式机器学习。由于数据的积累速度已经远远高于计算机的发展速度，因此在很多场景下，一台机器根本无法处理所有数据。为了能在这样的大数据上使用机器学习模型，需要将复杂的模型训练分解到由多台机器组成的计算集

[26] 非开源工具的问题可能更多，只是因为无法看到代码，所以问题被掩盖了。由于数据科学模型涉及很复杂的数学运算和工程实现，而且谁也无法保证自己写的代码没有 bug，因此，开源可能是验证模型实现是否正确的最佳方案。这也是开源在数据科学领域极其流行的原因之一。

群上。虽然表面上，模型的训练无法分解，但借助梯度下降法和 MapReduce 计算框架可以实现这一点。

在实际的大数据应用中，除了上面提到的分布式模型训练，我们还常常借助集群的力量，并行地训练大量“小”模型，比如为了求解最优超参数，可以在集群上并行地执行网点搜寻算法。

这两种应用场景，开源工具 Spark 都能很好地解决。对于第一种应用场景，它提供了封装好的分布式机器学习算法库；而对于第二种应用场景，Spark 有很优秀的任务调度引擎，可以很方便地在集群上分发模型训练。但由于分布式计算的复杂性，Spark 提供的算法库并不是很成熟，需要在使用时，根据实际情况调整模型的工程实现。事实上，这个问题并不是 Spark 所特有的，其他开源工具也都存在这样或那样的缺陷和 bug，需要在使用时多加小心。

本章只介绍了有关分布式机器学习和 Spark 的基本内容。想要了解更多的读者可参考其他书籍，比如 Matei Zaharia 等人编写的 *Learning Spark* 和 Sean Owen 等人编写的 *Advanced Analytics with Spark*。

第12章

神经网络：模拟人的大脑

By far the greatest danger of Artificial Intelligence is that people conclude too early that they understand it.（到目前为止，人工智能最大的威胁是人类自以为了解它。）

——*Eliezer Yudkowsky*

12.1 神经元

12.2 神经网络

12.3 反向传播算法

12.4 提高神经网络的学习效率

12.5 本章小结

在之前的章节中，我们讨论了很多不同类型的模型，它们大致可以分为两类：一类是比较注重模型可解释性的传统统计模型，比如第 4 章中的线性回归和第 5 章里的逻辑回归；另一类是侧重于从结构上“模仿”数据的机器学习模型，比如第 8、9 章中的监督式学习和第 10 章中的非监督式学习。

这些模型虽然在结构和形态上千差万别，但它们有一个共同的建模理念，就是首先对数据做假设，然后根据这些假设进行数学推导，并最终得到模型的公式。其中最核心的部分就是模型的假设，它直接决定了模型的适用范围，也是模型效果的保障。这些模型不但能对未知数据做预测，还能帮助我们去理解数据之间的相关关系。

本章和下一章将讨论的是一种全新的建模理念，它并不关心模型的假设以及相应的数学推导，也就是说它并不关心模型的可解释性。这个理念的目的是借鉴仿生学[1]的思路，利用计算机和数学模型去模拟人的大脑，因此，其中最核心的内容就是工程实现。从模型角度来讲，这种理念的建模起点是最终的模型公式，而模型效果的保障是，这样的模型能在一定程度上模仿大脑，而大脑是人类智能的基础。

这种理念下设计出来的模型有很多酷炫的名字，比如神经网络、人工智能以及深度学习等。这类模型虽然难以理解或者更准确地说，到目前为止人类还无法理解，但在某些特定应用场景里的预测效果却出奇得好，因此也常常引起争论。一部分人认为，目前的人工智能热只是一个泡沫，整个学科并没有实质性的突破；另一部分人认为人工智能已经在突破的前夜，在不远的未来，它将给人类带来巨大的便利；还有一部分人认为人工智能是极其危险的东西，我们正在创造一种新的具有智能的“生命”，也许在不远的未来，这种人造的智能会统治地球并最终毁灭人类，就像很多科幻电影里的情节那样。

以上这 3 种观点都有其道理[2]，本书并不打算加入这种宏观议题争论，而是采取中立立场讨论相关的技术细节和发展趋势。相信读者通过本书了解了人工智能的基础知识后，会对上面的话题有自己的观点。

---

[1] 仿生学（bionics）是模仿生物的特殊本领的一门科学，它在了解生物结构和功能原理的基础上，来研制新的机械和新的技术。以上简介参考自维基百科。

[2] 这 3 种观点的论据和逻辑超出了本书的讨论范围，在此就不做展开，仅列举它们背后的权威支持者。

机器学习领域重要的学者迈克尔·I.乔丹（Michael I. Jordan）教授就持第一种观点，他认为我们离接近人类水平的人工智能还很远。虽然在某些领域，可以用神经网络来“伪造”智能，但理智来说，这并不是智能。

另一位很知名的学者吴恩达（Andrew Ng，他是迈克尔·I.乔丹的学生）以及企业家扎克伯格（Mark Zuckerberg）持第二种观点。他们对人工智能的发展表示乐观，主张人工智能是一场新的工业革命，将会像电力一样改变工业以及人类的生活。

来自业界的比尔·盖茨（Bill Gates）和伊隆·马斯克（Elon Musk）则持第三种观点，他们认为虽然现阶段人工智能并没有表现出直接的危害，但按照现在的发展速度，在不远的将来（5 年或者 10 年之内），我们将直接面对人工智能带来的威胁。

12.1 神经元

由于神经网络的模拟对象是人的大脑，那么在讨论具体的模型之前，我们有必要先从生物学的角度来看看人的大脑有哪些特性。

根据生物学的研究，人脑的计算单元是神经元（neuron）。它能根据环境变化做出反应，再将信息给其他的神经元。在人脑中，大约有 860 亿个神经元，它们相互联结构成了极其复杂的神经系统，而后者正是人类智慧的物质基础。因此遵循人脑的生物结构，我们首先需要搭建模型来模拟人的神经元。

12.1.1 神经元模型

如图 12-1 所示，一个典型的神经元由 4 个部分组成。

- 树突：一个神经元有若干个树突，它们能接收来自其他神经元的信号，并将信号传递给细胞体。
- 细胞体：细胞体是神经元的核心，它把各个树突传递过来的信号加总起来，得到一个总的刺激信号。
- 轴突：当细胞体内的刺激信号超过一定阈值之后，神经元的轴突会对外发送信号。
- 突触：该神经元发送的信号（若有）将由突触向其他神经元或人体内的其他组织（对神经信号做出反应的组织）传递。需要注意的是，神经元通常有多个突触，但它们传递的信号都是一样的。

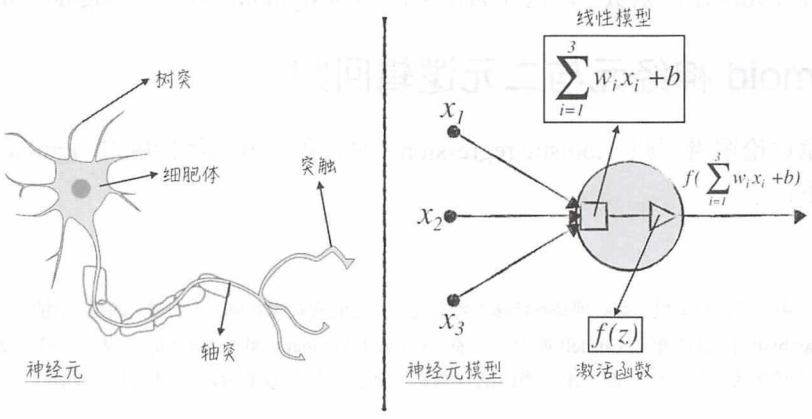


图 12-1

将上述的神经元结构抽象成数学概念，可以得到如图 12-1 所示的神经元模型。

- 模型的输入是数据里的自变量，比如图中的x_1, x_2, x_3。它们用圆点表示，对应着神经元里的树突。
- 接收输入变量的是一个线性模型，在图中用正方形表示。这个线性模型对应着神经元的细胞体。值得注意的是，对于神经元中的线性模型，我们将模型中的权重项和截距项特意分开，用w_i表示权重，用b表示截距[3]。
- 接下来是一个非线性的激活函数f（activation function），它将控制是否对外发送信号，在图中用三角形表示，对应这神经元里的轴突。在神经网络领域，常常用一个圆圈来概括地表示线性模型和激活函数，并不将两者分开，在本章和第 13 章的讨论中，我们将沿用这一记号。
- 将模型的各个部分联结起来得到最后的输出$f(\sum_i w_i x_i + b)$，这个值将传递给下一个神经元模型，在图中用箭头表示，对应着神经元里的突触。值得注意的是，一个神经元可以有多个输出箭头，但它们所输出的值都是一样的。

在神经元模型中，非线性的激活函数f是整个模型的核心。在最初的神经元模型中[4]，f的定义是非常直观的，当函数的自变量大于某个阈值时，则等于 1，否则等于 0。具体的公式如下：

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leqslant 0 \end{cases} \tag{12-1}$$

这个模型在学术上被称为感知器（perceptron），它可被用来解决二元分类问题（因为模型的输出是 0 或 1）。感知器虽然在某种程度上模拟了神经元里轴突的行为，但处理方式有些太过粗糙了，因为在生物学上，神经元输出的是一个连续值而非离散值。这导致感知器的模型效果很一般。为了改进这一点，通常使用 sigmoid 函数（sigmoid function，也称为 S 函数）来作为神经元的激活函数[5]，这样的模型被称为 sigmoid 神经元（sigmoid neuron）。

12.1.2 Sigmoid 神经元与二元逻辑回归

在第 5 章讨论逻辑回归（logistic regression）时，我们就已经接触过 sigmoid 函数了，具体的公式如下：

---

[3] 在神经网络中，线性模型里的截距项是有特殊生物含义的，它通常对应着神经元的激活阈值，因此需要单独处理它。

[4] Frank Rosenblatt 于 1957 年在 Cornell 航空实验室（Cornell Aeronautical Laboratory）设计了第一款人工神经网络。这个最初版的神经网络其实是一台机器：由于当时的计算机还处在比较初级的阶段，因此专门设计了一台机器来实现这个模型。

[5] 事实上基于工程实现上面的考虑，目前在实际应用中很少会使用 sigmoid 函数作为激活函数，具体的细节请参考 12.4.2 节。

$$S(z) = {}^{1}/_{(1 + e^{-z})} \tag{12-2}$$

使用 sigmoid 函数作为神经元里的激活函数有两大好处。从实用的角度来讲，sigmoid 函数能将任意的实数值映射到$(0,1)$区间，当公式（12-2）中的变量z是很大的负数时，函数值接近 0；当变量z是很大的正数时，函数值接近 1。这个特性在神经元上也能找到很好的解释：函数值接近 0 表示神经元没被激活，而函数值接近 1 表示神经元完全被激活。

从理论的角度来讲，sigmoid 函数模拟了两种效应的相互竞争：假设正效应和负效应都和自变量$\boldsymbol{X} = (x_1, x_2, \ldots, x_k)$是近似线性关系。具体的公式如下，其中，$Y^*$表示正效应，$Y^{\sim}$表示负效应，$\boldsymbol{W_i} = (w_{i,1}, w_{i,2}, \ldots, w_{i,k})$和$b_i$是模型参数，$\theta$和$\tau$是服从正态分布的随机干扰项。

$$\begin{aligned} Y^* &= \boldsymbol{X}\boldsymbol{W}_1^{\mathrm{T}} + b_1 + \theta \\ Y^{\sim} &= \boldsymbol{X}\boldsymbol{W}_2^{\mathrm{T}} + b_2 + \tau \end{aligned} \tag{12-3}$$

根据 5.1.2 节和 5.1.3 节中的讨论，正效应大于负效应的概率可由一个 sigmoid 函数来近似，如公式（12-4）所示。

$$P(Y^* - Y^{\sim}) \approx S(\boldsymbol{X}\boldsymbol{W}^{\mathrm{T}} + b) \tag{12-4}$$

因此在神经元模型里使用 sigmoid 函数，就相当于给神经元的输出赋予了概率意义，这使得模型的理论基础更加扎实，也使得模型能被用于解决二元分类问题，比如当 sigmoid 神经元的输出大于 0.5 时，则预测类别为 1，否则预测类别为 0。值得注意的是，在这种情况下，sigmoid 神经元其实就是二元逻辑回归模型，如图 12-2 所示。

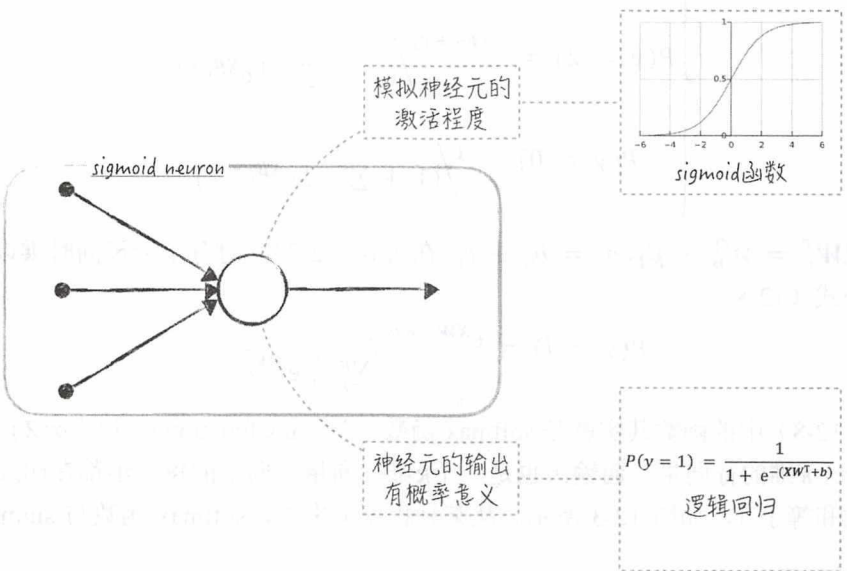


图 12-2

书籍是人类进步的阶梯

12.1.3 Softmax 函数与多元逻辑回归

为了在之后的章节中更深入地讨论神经网络，本节将介绍在这个领域里很重要的 softmax 函数，它常被用来定义神经网络的损失函数（针对分类问题）。

根据 5.1.3 节中的讨论，二元逻辑回归的模型公式可以写为如下的形式：

$$P(y=1)={}^{1}\!/_{[1+\mathrm{e}^{-(\boldsymbol{X}\boldsymbol{W}^{\mathrm{T}}+b)}]} \tag{12-5}$$

在公式（12-5）中，对分子、分母同时乘以$\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_1^{\mathrm{T}}+b_1}$，得到公式（12-6），其中，$\boldsymbol{W}_0=\boldsymbol{W}_1-\boldsymbol{W}$；$b_0=b_1-b$。

$$\begin{aligned}P(y=1)&={}^{\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_1^{\mathrm{T}}+b_1}}\!/_{(\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_1^{\mathrm{T}}+b_1}+\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_0^{\mathrm{T}}+b_0})}\\P(y=0)&={}^{\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_0^{\mathrm{T}}+b_0}}\!/_{(\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_1^{\mathrm{T}}+b_1}+\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_0^{\mathrm{T}}+b_0})}\end{aligned} \tag{12-6}$$

事实上，多元逻辑回归的模型公式也可以写成类似的形式。具体地，根据 5.4.1 节中的讨论，假设分类问题有k个类，分别记为$0,1,\ldots,k-1$，则多元逻辑回归的模型可以表示为如下的形式[6]。

$$\begin{cases}P(y=1)={}^{\mathrm{e}^{\boldsymbol{X}\boldsymbol{\beta}_1+c_1}}\!/_{(1+\sum_{j=1}^{k-1}\mathrm{e}^{\boldsymbol{X}\boldsymbol{\beta}_j+c_j})}\\P(y=2)={}^{\mathrm{e}^{\boldsymbol{X}\boldsymbol{\beta}_2+c_2}}\!/_{(1+\sum_{j=1}^{k-1}\mathrm{e}^{\boldsymbol{X}\boldsymbol{\beta}_j+c_j})}\\\cdots\\P(y=0)={}^{1}\!/_{(1+\sum_{j=1}^{k-1}\mathrm{e}^{\boldsymbol{X}\boldsymbol{\beta}_j+c_j})}\end{cases} \tag{12-7}$$

不妨记$\boldsymbol{W}_i^{\mathrm{T}}=\boldsymbol{W}_0^{\mathrm{T}}+\boldsymbol{\beta}_i, b_i=b_0+c_i$。在公式(12-7)中对分子分母同时乘以$\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_0^{\mathrm{T}}+b_0}$，可以得到公式（12-8）。

$$P(y=l)={}^{\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_l^{\mathrm{T}}+b_l}}\!/_{\sum_{j=0}^{k-1}\mathrm{e}^{\boldsymbol{X}\boldsymbol{W}_j^{\mathrm{T}}+b_j}} \tag{12-8}$$

公式（12-8）中的函数其实就是 softmax 函数（softmax function），记为$\sigma(\boldsymbol{Z})$。这个函数的输入是一个k维的行向量，而输出也是一个k维行向量，向量的每一维都在$(0,1)$区间中，而且加总的和等于 1，如图 12-3 所示。从某种程度上来讲，softmax 函数与 sigmoid 函数非

[6] 本章中的公式（12-7）与 5.4.1 节中的公式（5-28）稍有不同。在公式（12-7）中，我们将截距项c_i单独列了出来（而在公式（5-28）中，截距项和权重项合在一起记为$\boldsymbol{\beta}_i$）。这样书写公式主要是为了和神经网络领域里的其他文献保持一致。

书籍是人类进步的阶梯

常类似，它们都能将任意的实数“压缩”到$(0,1)$区间。

$$Z=(z_0,z_1,\ldots z_{k-1}) \longrightarrow \boxed{\text{softmax function}} \longrightarrow \left(\frac{e^{z_0}}{\sum e^{z_j}},\frac{e^{z_1}}{\sum e^{z_j}},\ldots\frac{e^{z_{k-1}}}{\sum e^{z_j}}\right)$$

图 12-3

在 softmax 函数的基础上，可以将逻辑回归转换成图的形式，这样可以更直观地在神经网络里使用这个模型（在机器学习领域，复杂的神经网络常被表示为图）。以二元逻辑回归为例，得到的图像如图 12-4 所示。图中的方块表示线性模型，与图 12-1 中的方块表示的含义是一样的。另外值得注意的是，图 12-4 所表示的模型与图 12-2 中的 sigmoid 神经元模型是一致的，只是图 12-4 可以很轻松地扩展到多元分类问题（增加图中方块的数目），而图 12-2 中的模型只能解决二元分类问题。

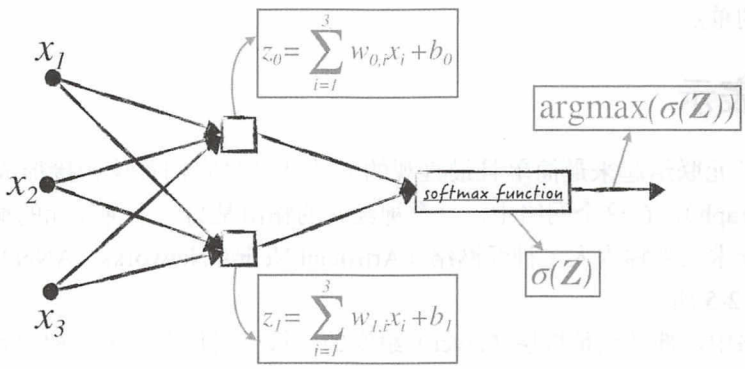


图 12-4

另外，借助 softmax 函数，逻辑回归模型的损失函数L可以被改写为更简洁的形式，如公式（12-9）所示（有关细节可参考 5.4.1 节）。

$$L = -\sum_i \sum_{j=0}^{k-1} 1_{\{y=j\}} \ln\left(e^{\boldsymbol{X}_i \boldsymbol{W}_j^{\mathrm{T}} + b_j} \Big/ \sum_{l=0}^{k-1} e^{\boldsymbol{X}_i \boldsymbol{W}_l^{\mathrm{T}} + b_l}\right) \tag{12-9}$$

那么，对于k元分类问题，假设第i个数据的类别是t，用一个k维的行向量$\boldsymbol{\theta}_i = (\theta_{i,0},\theta_{i,1},\ldots,\theta_{i,k-1})$来表示它的类别[7]：这个行向量的第$t$个维度等于 1，即$\theta_{i,t} = 1$，其他维度等于 0，即$\theta_{i,j} = 0, j \neq t$。基于此，逻辑回归在这一个数据点上的损失可以写成 softmax

[7] 这种处理方法在学术上被称为 One-Hot Encoding（独热编码）。

书籍是人类进步的阶梯

函数与行向量$\boldsymbol{\theta}_i$矩阵乘法的形式（也可以认为是向量内积的形式），如公式（12-10）所示，其中$\boldsymbol{Z}_i = (\boldsymbol{X}_i\boldsymbol{W}_0^{\mathrm{T}} + b_0, \ldots, \boldsymbol{X}_i\boldsymbol{W}_{k-1}^{\mathrm{T}} + b_{k-1})$是一个$k$维的行向量。

$$L_i = -\boldsymbol{\theta}_i \ln \sigma(\boldsymbol{Z}_i)^{\mathrm{T}} \tag{12-10}$$

类似地，整个模型的损失函数也可以写为矩阵乘法的形式（因为$L = \sum_i L_i$），这样的形式对神经网络的工程实现十分有用，在之后的章节里会经常遇到基于它的代码实现。

12.2 神经网络

在上一节中，我们讨论了如何为神经元搭建模型。虽然搭建模型的过程并不复杂，但得到的神经元模型也没有太多的新意，比如使用 sigmoid 函数作为激活函数，则得到的神经元模型就是逻辑回归。

在人体中，单个神经元能做的事情非常有限，但多个神经元相互交织在一起就组成了人类强大的神经系统。这启发我们需要将多个神经元模型联结起来组成复杂的神经网络，而这正是本节讨论的重点。

12.2.1 图形表示

将多个神经元联结起来最简单且最直观的方法就是将它们首尾相接形成一个没有环的网络（acyclic graph），在这个网络中，一个神经元的输出是另一个神经元的输入。这种类型的神经网络在学术上被称为人工神经网络（Artificial Neural Networks，ANN）[8]，它的模型拓扑结构如图 12-5 所示。

在神经网络中，神经元是按层（layer）组织的。每一层包含若干个神经元，层内部的神经元是相互独立的，也就是说它们之间并不相连；但相邻的两层之间是全连接的（fully-connected），也就是说任意两个神经元都是直接相连的（当然前提是这两个神经元分别来自相邻的两层）。

神经网络中不同的层按功能分为 3 类，分别是输入层（input layer）、隐藏层（hidden layer）以及输出层（output layer）。

（1）神经网络里只有一个输入层，其中的元素在图中标记 1 里用黑色的点表示：一个黑点表示一个模型的输入，也就是训练数据里的一个自变量，若训练数据有k个特征，则输入层里一定有相应的k个点。在模型中，输入层对数据不做任何处理，只负责将信息传递给后面的隐藏层（若网络里没有隐藏层，则将信息直接传递给输出层，这时候，神经网络就是如

[8] 在有些文献中，这样的神经网络又被称为多层感知器（Multi-Layer Perceptron，MLP）。

书籍是人类进步的阶梯

图 12-4 所示的逻辑回归）。

（2）神经网络可以有多个隐藏层，比如图 12-5 中就有两个隐藏层。隐藏层里的元素就是之前讨论的神经元模型（更具体一点，它们都是 sigmoid 神经元），因此在图中标记 1 里用圆圈表示。值得注意的是，一个圆圈包含了线性模型以及激活函数。在多层感知器里，隐藏层的作用是传输并分析数据。

（3）神经网络里只有一个输出层。输出层里的元素与隐藏层里的不同，它只包含线性模型，因此在图中标记 1 里用方块表示。虽然输出层的名字里有“输出”二字，但它并不是模型的最终输出，与图 12-4 中的逻辑回归类似，输出层的结果经过 softmax 函数处理后，才能得到最终的模型结果（如果该神经网络解决的是分类问题）。

正如上面讨论的，不同层里的元素是不一样的，但对于神经网络领域，大家约定俗成地都用圆圈来表示它们，也将它们统称为神经元（虽然这样很容易给初学者造成误解），并将最后一步的 softmax 函数省略掉，如图 12-5 中标记 2 所示。

（1）图 12-5 中的标记 1 和标记 2 表示的同样的神经网络，其中标记 2 中的图示在神经网络领域是通用的，因此在之后的章节中我们将沿用这样的记号。

（2）对于神经网络，通常会以它所拥有的层数来命名，但这时输入层是不计算在内的，比如图 12-5 中的模型被称为 3-层神经网络（3-layer neural networks）。当然这样的命名并不能唯一地标识一个神经网络，比如将图 12-5 中的隐藏层 1 增加到 10 个神经元，得到的仍然是一个 3-层神经网络。

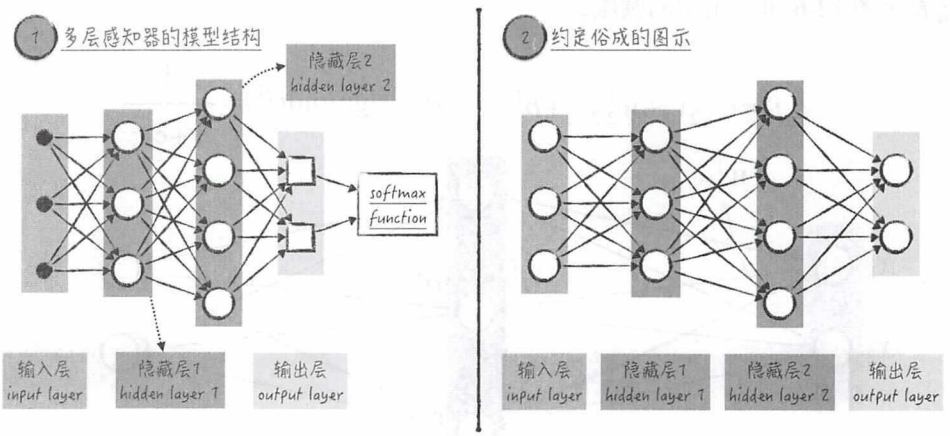


图 12-5

12.2.2 数学基础

神经网络在图形上的表示是比较直观的，但它所代表的数学公式却是极其复杂的。为了

书籍是人类进步的阶梯

更深入地理解神经网络，下面将讨论图形背后的数学基础。

为了讨论方便，我们来看一个非常简单的 2-层神经网络（假设这个神经网络用于解决分类问题），如图 12-6 所示。

这个网络的输入层有两个圆圈，表示模型所用的两个自变量，分别用x_1, x_2表示。

隐藏层里的圆圈表示神经元模型，它包含两个部分：线性模型和激活函数。

（1）对于第l层（从左到右编号）里的第m个神经元（从上到下编号），用记号i_m^l表示相应的线性模型输出（圆圈输入的第一层加工），用记号o_m^l表示激活函数的输出（也是圆圈的输出）。由于这个神经网络里都是 sigmoid 神经元，因此两者之间的关系如公式（12-11）所示。

$$o_m^l = 1/(1 + \mathrm{e}^{-i_m^l}) \tag{12-11}$$

（2）图中圆圈之间的箭头表示线性模型中的权重（这是神经网络的一部分模型参数），用记号$w_{m,n}^l$表示从第$l-1$层里第m个圆圈到第l层里第n个圆圈的箭头，比如图 12-6 中的$w_{1,2}^1$和$w_{3,1}^2$。值得注意的是，为了书写方便，我们将输入层记为第 0 层。

（3）除了权重之外，线性模型里还有一个截距项（这是神经网络的另一部分模型参数），用记号b_m^l表示第l层里第m个圆圈所对应的截距。结合上面的记号，圆圈里线性模型的输出公式如下。值得注意的是，o_m^0表示输入层里第m个自变量，也就是图中的x_m。

$$i_n^l = \sum_m w_{m,n}^l \, o_m^{l-1} + b_n^l \tag{12-12}$$

输出层的圆圈只表示线性模型，虽然圆圈里并没有激活函数，但为了书写简单，依然用记号o_m^2表示图 12-6 里输出层的圆圈。

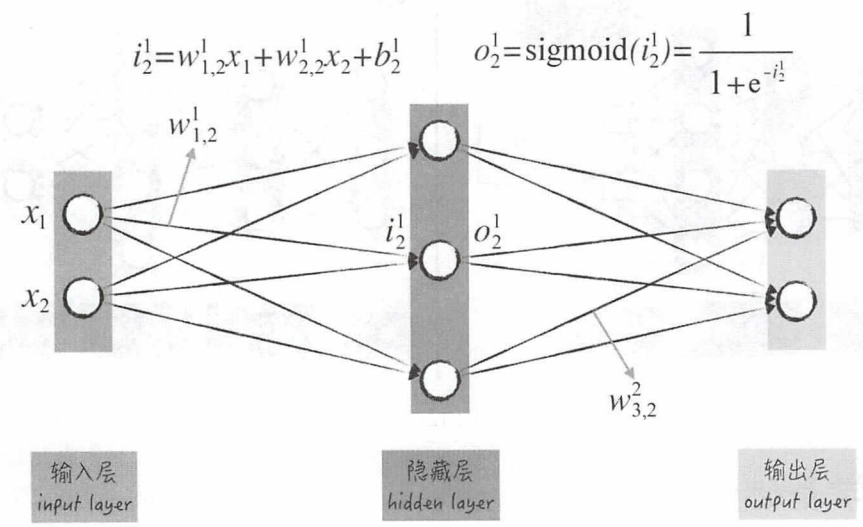


图 12-6

书籍是人类进步的阶梯

（1）与隐藏层有所不同，在输出层里，我们有$i_m^l = o_m^l$（i_m^l的计算公式与隐藏层中的一模一样，在此不再赘述）。

（2）针对分类问题，相应的损失函数如公式（12-13）所示。其中σ为 12.1.3 节中讨论过的 softmax 函数；$\boldsymbol{Z_i} = (o_1^2, o_2^2)_i$表示针对第$i$个数据的神经网络输出；$\boldsymbol{\theta_i}$表示的第$i$个数据的类别，如果是类别 0，则$\boldsymbol{\theta_i} = (1,0)$，否则$\boldsymbol{\theta_i} = (0,1)$。

$$L = \sum_i L_i = -\sum_i \boldsymbol{\theta_i} \ln \sigma(\boldsymbol{Z_i})^{\mathrm{T}} \tag{12-13}$$

将上面的讨论总结一下，神经网络的模型参数分为两类：一类是线性模型的权重$w_{m,n}^l$，在图 12-6 中，一共有$3 \times 2 + 2 \times 3 = 12$个这样的参数；一类是线性模型的截距项$b_m^l$，在图 12-6 中，一共有 5 个这样的参数。由此可见，神经网络的模型参数是很多的，而且与其他模型有所区别的是，它的模型参数是明显分层的。也就是说，从数学上来看，神经网络的模型参数是前后依赖的。这使得神经网络的训练变得十分复杂，需要使用比较特殊的算法来估算模型参数。这部分内容将在第 12.3 节仔细讨论。

12.2.3 分类例子[9]

上面的讨论重点是神经网络的理论知识。现在来看一个实际的例子，如何利用神经网络解决分类问题。为了更好地展示神经网络的特点，我们在这个示例中并不划分训练集和测试集。

分类是机器学习最常见的应用之一，之前的章节也讨论过很多解决分类问题的机器学习模型，比如第 5 章讨论的逻辑回归和第 8 章讨论的支持向量学习机等。但这些模型最大的局限性是它们都有比较明确的适用范围，如果训练数据符合模型的假设，则分类效果很好。否则，分类的效果就会很差。

比如图 12-7 中展示了 4 种不同分布类型的数据。具体来说，数据里有两个自变量，分别对应着坐标系的横纵轴；数据分为两类，在图中用三角形表示类别 0，用圆点表示类别 1。如果使用逻辑回归对数据进行分类，只有图中标记 1 中的模型效果较好（图中的灰色区域里，模型的预测结果是类别 0；白色区域里，模型的预测结果是类别 1），因为在已知类别的情况下，数据服从正态分布（不同类别，分布的中心不同），符合逻辑回归的模型假设。对于标记 2、3、4 中的数据，由于类别与自变量之间的关系是非线性的，如果想取得比较好的分类效果，则需要其他的建模技巧。比如先使用核函数对数据进行升维，再使用支持向量学习机进行分类。

---

[9] 例子参考自 GitHub 上的开源项目 tensorflow/playground。完整的实现请请参考随书配套的代码/ch12-ann/classification_example.py。

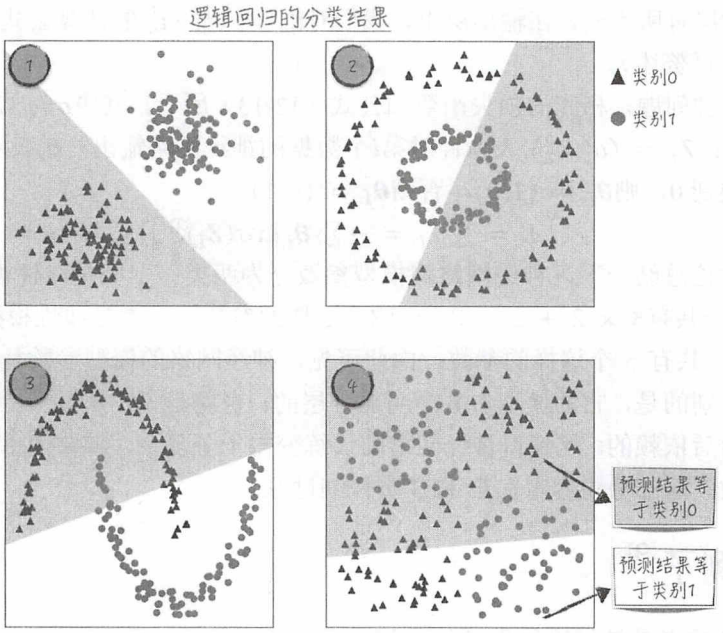


图 12-7

这样的建模方法是比较辛苦的，要求搭建模型的数据科学家对不同模型的假设以及优缺点有比较深刻的理解。但如果使用神经网络对数据进行分类，则整个建模过程就比较轻松了，只需设计神经网络的形状（包括神经网络的层数以及每一层里的神经元个数），然后将数据输入给模型即可。

在这个例子中，使用的神经网络如图 12-8 所示，是一个 3-层的全连接神经网络。

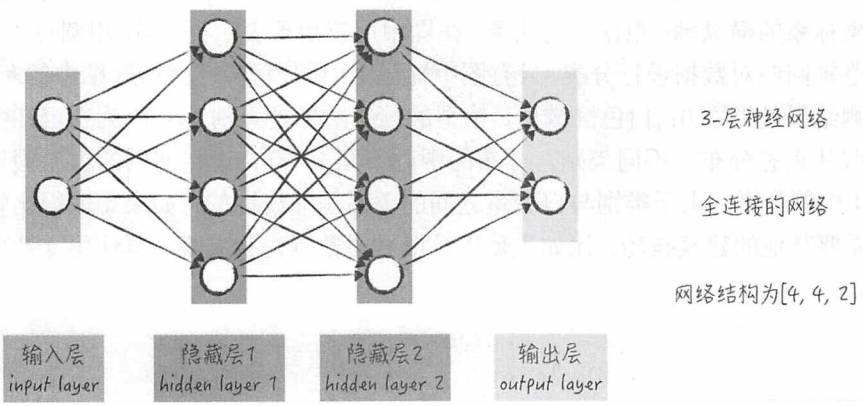


图 12-8

使用这个神经网络对数据进行分类，得到的结果如图 12-9 所示，可以看到同一个神经网络（结构相同，但具体的模型参数是不同的）对 4 种不同分布类型的数据都能较好地进行分类。

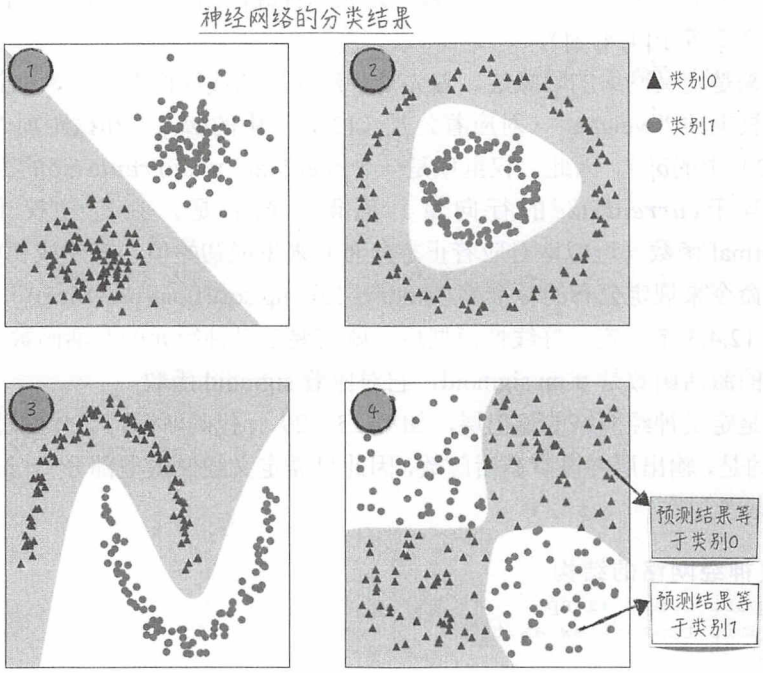


图 12-9

12.2.4 代码实现[10]

本节将讨论如何借助第三方库 TensorFlow（有关 TensorFlow 的基础知识请参考第 6.3 节）来实现神经网络。

第一步是定义神经网络的结构，如程序清单 12-1 所示。

（1）我们使用类（class）来实现神经网络，如第 4 行代码所示。在 Python 的类中可以定义相应的函数，但在类中，函数的定义与普通函数的定义有所不同，它的参数个数必须大于 1，且第一个参数表示类本身，如第 7 行代码里的“self”变量。但在调用这个函数时，却不需要“手动”地传入这个参数，Python 会自动地进行参数传递，比如 defineANN 函数

[10] 为了避免陷入过多的细节，正文中的程序清单省略了部分代码，比如记录模型日志的代码。完整的实现请参考随书配套的代码/ch12-ann/mlp.py。

的调用方式是“defineANN()”。

（2）在 ANN 类中，“self.input”对应着训练数据里的自变量（它的类型是 tf.placeholder），如第 12 行代码所示，“self.input.shape[1].value”表示输入层的神经元个数（针对如图 12-8 的神经网络，这个值等于 2）。而“self.size”是表示神经网络结构的数组（针对如图 12-8 的神经网络，这个值等于[4, 4, 2]）。

（3）接下来是定义网络的隐藏层。首先是神经元里的线性模型部分，如第 18～21 行代码所示，定义权重项“weights”（对应着公式（12-12）中的$w_{m,n}^{l}$）和截距项“biases”（对应着公式（12-12）中的b_{n}^{l}）。因此，权重项是一个$prevSize \times currentSize$的矩阵，而截距项是一个维度等于$currentSize$的行向量。值得注意的是，在定义权重项时，使用 tf.truncated_normal 函数（近似地对应着正态分布）来生成初始值，在生成初始值的过程中，我们用如下的命令来规定分布的标准差“stddev=1.0 / np.sqrt(float(prevSize))”，这样操作的具体原因请参考 12.4.3 节。定义好线性模型后，就需要定义神经元的激活函数，如第 22 行代码所示，使用的激活函数是 tf.nn.sigmoid，它对应着 sigmoid 函数。

（4）最后是定义神经网络的输出层，如第 25～29 行代码所示。具体的过程和隐藏层类似，唯一不同的是，输出层并没有激活函数，因此只需定义线性模型部分“tf.matmul(prevOut, weights) + biases”。

程序清单 12-1 定义神经网络的结构

```
 1 | import numpy as np
 2 | import tensorflow as tf
 3 |
 4 | class ANN(object):
 5 |     # 省略掉其他部分
 6 |
 7 |     def defineANN(self):
 8 |         """
 9 |         定义神经网络的结构
10 |         """
11 |         # self.input 是训练数据里自变量
12 |         prevSize = self.input.shape[1].value
13 |         prevOut = self.input
14 |         # self.size 是神经网络的结构，也就是每一层的神经元个数
15 |         size = self.size
16 |         # 定义隐藏层
17 |         for currentSize in size[:-1]:
18 |             weights = tf.Variable(
19 |                 tf.truncated_normal([prevSize, currentSize],
20 |                     stddev=1.0 / np.sqrt(float(prevSize))))
21 |             biases = tf.Variable(tf.zeros([currentSize]))
22 |             prevOut = tf.nn.sigmoid(tf.matmul(prevOut, weights) + biases)
23 |             prevSize = currentSize
24 |         # 定义输出层
25 |         weights = tf.Variable(
```

```
26 |             tf.truncated_normal([prevSize, size[-1]],
27 |                 stddev=1.0 / np.sqrt(float(prevSize))))
28 |         biases = tf.Variable(tf.zeros([size[-1]]))
29 |         self.out = tf.matmul(prevOut, weights) + biases
30 |         return self
```

第二步是定义神经网络的损失函数，如程序清单 12-2 所示。

（1）在 ANN 类中，“self.label”对应着训练数据里的标签变量（它的类型是 tf.placeholder）。值得注意的是，这里用到的标签变量是使用 One-Hot Encoding（独热编码）处理过的。比如针对图 12-9 中的数据，每个数据的标签变量是二维的行向量，用$[1,0]$表示类别 0，用$[0,1]$表示类别 1。

（2）在 ANN 类中，“self.out”对应着神经网络的输出层，具体的定义如程序清单 12-1 中的第 29 行代码所示。

（3）根据 12.1.3 节和 12.2.2 节中的讨论结果，神经网络的单点损失如公式（12-10）所示。对应的实现如第 9、10 行代码所示，其中，“self.out”对应着公式里的变量Z_i。

（4）模型的整体损失等于所有单点损失之和，相应的实现如第 12 行代码所示。

程序清单 12-2 定义神经网络的结构

```
 1 | class ANN(object):
 2 |     # 省略掉其他部分
 3 |
 4 |     def defineLoss(self):
 5 |         """
 6 |         定义神经网络的损失函数
 7 |         """
 8 |         # 定义单点损失，self.label 是训练数据里的标签变量
 9 |         loss = tf.nn.softmax_cross_entropy_with_logits(
10 |             labels=self.label, logits=self.out, name="loss")
11 |         # 定义整体损失
12 |         self.loss = tf.reduce_mean(loss, name="average_loss")
13 |         return self
```

第三步是训练神经网络，如程序清单 12-3 所示。

（1）从理论上来讲，训练神经网络的算法是之后将讨论的反向传播算法（请参考第 12.3 节），这个算法的基础是第 6 章中讨论的随机梯度下降法（stochastic gradient descent）。由于 TensorFlow 已经将整个算法包装好了，因此训练神经网络的代码与随机梯度下降法的代码（细节内容请参考第 6.4 节）是一致的，如第 8～23 行代码所示。限于篇幅，实现的具体细节在此就不再重复了。

（2）如果将训练过程的模型损失（随训练轮次的变化曲线）记录下来，可以得到如图 12-10 所示的图像，其中曲线的标记对应着训练数据的标记。从图中的结果可以看到，对于不同类型的数据，模型损失函数的变化曲线是不一样的。对于比较难训练的数据（标记 3），模型的损失经历了一个很漫长的训练瓶颈期。也就是说，虽然模型并没有达到收敛状态，但

在较长的训练周期里，模型效果几乎没有提升。这种现象其实是神经网络研究领域里最大的难点，它使得神经网络的训练（特别是层数较多深度神经网络）变得极其困难，一方面瓶颈期会使模型的训练变得非常漫长；另一方面，在实际应用中，当模型损失不再大幅变动时，我们很难判断这是因为模型到达了收敛状态还是因为模型进入了瓶颈期[11]。引起瓶颈期这种现象的原因有很多，我们将在 12.4 节中重点讨论这部分内容。

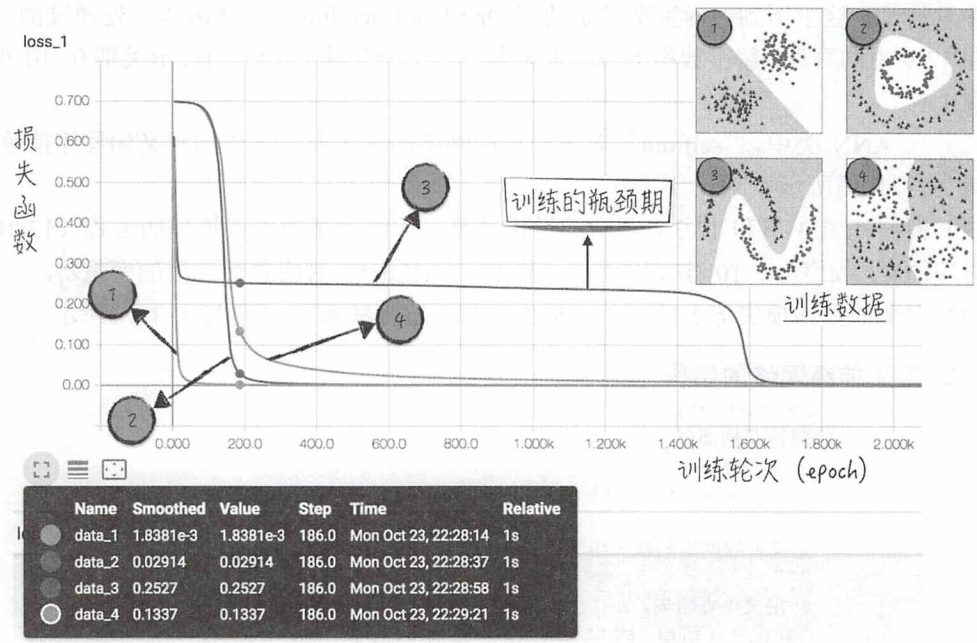


图 12-10

程序清单 12-3 训练模型

```
class ANN(object):
    # 省略掉其他部分

    def SGD(self, X, Y, learningRate, miniBatchFraction, epoch):
        """
        使用随机梯度下降法训练模型
        """
        method = tf.train.GradientDescentOptimizer(learningRate)
        optimizer= method.minimize(self.loss)
        batchSize = int(X.shape[0] * miniBatchFraction)
        batchNum = int(np.ceil(1 / miniBatchFraction))
```

[11] 虽然对于特定的应用场景，我们在数学上可以找到一些判断瓶颈期的依据，但从整体上来说并没有特别通用的办法，这一点也显示了人类对神经网络的理解是十分薄弱的。

```
12 |         sess = tf.Session()
13 |         init = tf.global_variables_initializer()
14 |         sess.run(init)
15 |         step = 0
16 |         while (step < epoch):
17 |             for i in range(batchNum):
18 |                 batchX = X[i * batchSize: (i + 1) * batchSize]
19 |                 batchY = Y[i * batchSize: (i + 1) * batchSize]
20 |                 sess.run([optimizer],
21 |                     feed_dict={self.input: batchX, self.label: batchY})
22 |             step += 1
23 |         self.sess = sess
24 |         return self
```

神经网络训练好之后，就可以使用它对未知数据做预测，如程序清单 12-4 所示。如 12.1.3 节中的公式（12-8）所示，对神经网络的输出层使用 softmax 函数，就可以得到每个类别的预测概率，具体的实现如第 9、10 行代码所示。

程序清单 12-4 对未知数据做预测

```
 1 | class ANN(object):
 2 |     # 省略掉其他部分
 3 |
 4 |     def predict_proba(self, X):
 5 |         """
 6 |         使用神经网络对未知数据进行预测
 7 |         """
 8 |         sess = self.sess
 9 |         pred = tf.nn.softmax(logits=self.out, name="pred")
10 |         prob = sess.run(pred, feed_dict={self.input: X})
11 |         return prob
```

12.2.5 模型的联结

在之前的章节中，我们曾多次讨论过模型的联结主义（connectionism）。这个理念的核心思想是通过网络的形式，将简单的模型组装成一个功能强大的模型，神经网络就是这一理念的极端代表。以 12.2.3 节中的模型为例，当使用一个逻辑回归对数据分类时，模型的效果并不理想。但如果将多个逻辑回归联结成网络[12]，则模型效果会得到极大的提升。

当然并不是所有模型联结成网络，模型效果都会提升。比如将神经元模型的激活函数设定为线性函数，则不管连接的网络如何复杂，最后得到的还是一个线性回归模型（因为从数学上来讲，线性函数的线性组合还是线性函数）。因此，虽然神经元的激活函数有很多种选择，但一定不能是线性函数。

从模型的结构上来看，神经网络可以被看成是多个线性回归模型的非线性叠加。如果将

[12] 根据 12.1 节中的讨论，当神经元模型使用 sigmoid 函数作为激活函数时，神经元模型就等同于逻辑回归。

神经网络想象成一个多层的奶油蛋糕，那么线性模型就是其中的蛋糕体，而非线性变换则是蛋糕体上面的奶油。

也可以反过来理解神经网络解决问题的思路：通过一层层的非线性变换将原本非线性的问题转换为近似线性的问题来解决。仔细想想，本书之前章节讨论的很多模型都有与之类似的模型结构和建模思路。这也是本书一直强调的一种观点，搭建模型的目的就是要将非线性问题转换为线性问题，因为从某种意义上来讲，到目前为止人类只能解决线性问题。

12.3 反向传播算法

在第 6 章中，我们讨论了如何利用随机梯度下降法来实现模型参数的估算。神经网络的参数估算也是使用这个方法，只是由于神经网络的模型结构十分复杂，导致损失函数对每个参数的偏导数计算非常繁琐，需要使用特殊的反向传播算法[13]（backpropagation，也称 BP 算法）。这个方法神经网络里最核心的技术，可以毫不夸张地说，它是整个领域的基石[14]。

为了加深读者对算法的理解，首先来回顾一下估算模型参数的随机梯度下降法。

12.3.1 随机梯度下降法回顾

对于一个搭建好的模型，不妨假设它的模型参数为$\boldsymbol{\beta} = (\beta_1, \beta_2, \ldots, \beta_k)$，损失函数为$L$。其中模型整体的损失可以写成模型在每一点的损失之和，如公式（12-14）所示，其中L_i表示模型在第i点的损失。

$$L(\boldsymbol{\beta}) = \sum_i L_i(\boldsymbol{\beta}) \tag{12-14}$$

数学上可以证明，如果使用公式（12-15）不停迭代，则模型参数会最终收敛到某个局部极值点。

$$\beta_j^{l+1} = \beta_j^l - \gamma \sum_i \frac{\partial L_i}{\partial \beta_j} \tag{12-15}$$

在上面的公式中，γ是学习速率（learning rate），表示每次迭代更新模型参数的力度；而$\sum_i \frac{\partial L_i}{\partial \beta_j}$表示模型损失函数对参数$\beta_j$的偏导数，如果使用所有的数据点来估算此偏导数，则对应的算法是梯度下降法，但这个算法在实际中并不常用；如果使用随机选取的部分数据点来

[13] 从数学上来看，反向传播算法并没有太多新颖之处，算法的核心就是如何工程化地利用链式法则来计算多元函数的偏导数。

[14] 虽然反向传播算法是神经网络的基石，但这个基石本身却是饱受质疑的。因为这个算法并没有生物学上的支持，这违背了神经网络的建模理念。

仅供非商业用途或交流学习使用

估算此偏导数，则对应的算法是随机梯度下降法。

从公式（12-15）中可以得到，若要使用随机梯度下降法来训练模型，则最核心的部分是计算损失函数的偏导数，这将是下一节讨论的内容。

12.3.2 数学推导

现在来推导反向传播算法的数学公式，也就是神经网络损失函数的偏导数。本节的推导过程比较繁琐，对数学细节不感兴趣的读者，可忽略计算过程，直接阅读后面的推导结果。

在下面的推导过程中，我们沿用 12.2.2 节中的数学记号。为了使计算过程更加容易理解，使用如图 12-11 所示的神经网络来展示所用的符号（这些内容在 12.2.2 节中已经讨论过了，这里只是再次回顾一下）：

- 神经网络的模型参数有两类，分别是权重项$w_{m,n}^{l}$和截距项b_{m}^{l}。其中$w_{m,n}^{l}$对应着从第$l-1$层里第m个圆圈到第l层里第n个圆圈的箭头，而b_{m}^{l}对应着第l层里第m个圆圈。
- 为了计算方便，还引入另外几个中间变量。用i_{n}^{l}表示第l层里第n个圆圈的线性模型输出；用o_{n}^{l}表示相应的激活函数输出，这两个中间变量满足如下的公式，其中f表示激活函数（神经网络里只有一种激活函数）。

$$
\begin{aligned}
i_{n}^{l} &= \sum_{m} w_{m,n}^{l} o_{m}^{l-1} + b_{n}^{l} \\
o_{n}^{l} &= f(i_{n}^{l})
\end{aligned}
\tag{12-16}
$$

- 神经网络的损失函数记为L。损失函数对中间变量i_{n}^{l}的偏导数记为δ_{n}^{l}，具体的如公式（12-17）所示：

$$
\delta_{n}^{l} = \frac{\partial L}{\partial i_{n}^{l}} \tag{12-17}
$$

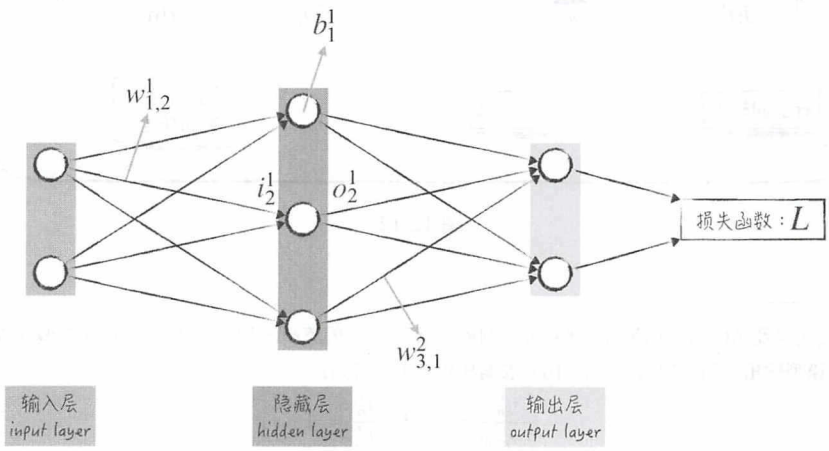


图 12-11

仅供非商业用途或交流学习使用

不妨假设神经网络一共有T层，根据公式（12-17）引入的偏导数δ_n^l和公式（12-16），可以很容易地得到模型参数的偏导数，如公式（12-18）所示[15]：

$$\frac{\partial L}{\partial w_{m,n}^l} = \frac{\partial L}{\partial i_n^l}\frac{\partial i_n^l}{\partial w_{m,n}^l} = \delta_n^l o_m^{l-1}$$

$$\frac{\partial L}{\partial b_n^l} = \frac{\partial L}{\partial i_n^l}\frac{\partial i_n^l}{\partial b_n^l} = \delta_n^l \tag{12-18}$$

在上面的公式中，只有δ_n^l还是未知的，现在推导它的公式，根据链式法则可以得到[16]：

$$\frac{\partial L}{\partial i_n^l} = \sum_m \frac{\partial L}{\partial i_m^{l+1}}\frac{\partial i_m^{l+1}}{\partial i_n^l} = \sum_m \frac{\partial L}{\partial i_m^{l+1}}\frac{\partial i_m^{l+1}}{\partial o_n^l}\frac{\partial o_n^l}{\partial i_n^l}$$

$$\delta_n^l = \sum_m \delta_m^{l+1} w_{n,m}^{l+1} f'(i_n^l) \tag{12-19}$$

公式（12-19）是一个迭代公式，它的迭代尽头是神经网络的输出层，也就是δ_n^T。由于在输出层，我们有$i_n^T = o_n^T$，因此可以得到公式（12-20）。需要注意的是，向量$(o_1^T, \ldots, o_n^T, \ldots)$其实就是损失函数的自变量，因此，公式（12-20）的右边就是损失函数梯度的第n个元素。

$$\delta_n^T = \frac{\partial L}{\partial i_n^T} = \frac{\partial L}{\partial o_n^T} \tag{12-20}$$

上面得到公式（12-18）～公式（12-20）就是反向传播算法，如图 12-12 所示。对于这一长串的公式，应该如何理解它们呢？从直观上来讲，损失函数L表示模型的预测错误，因此δ_n^T可以表示在输出层各个神经元所犯的错误，这是整个计算的起点。接下来的有关δ_n^l的迭代公式则表示把错误通过神经网络反向传播到每一个神经元。最后把神经元的错误分解到各个模型参数，而后者正是训练模型所需的梯度。

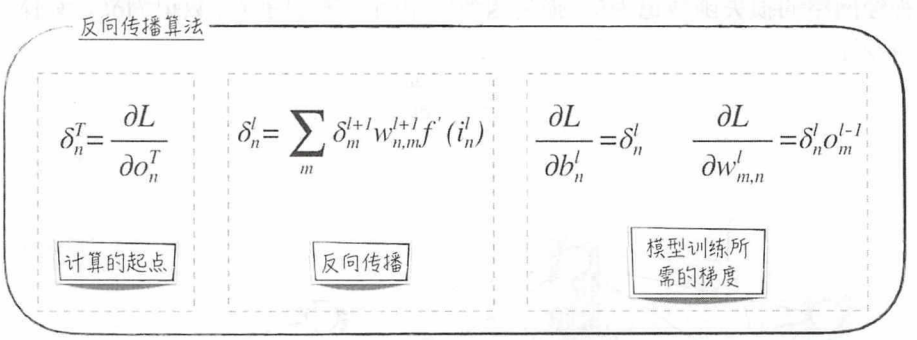


图 12-12

[15] 由于$w_{m,n}^l$只涉及第$l-1$层的第m个神经元（圆圈）以及第l层的第n个神经元，因此，在计算偏导数时并不涉及其他神经元的线性模型输出。另外对公式（12-16）求偏导数，可以得到：

$$\frac{\partial i_n^l}{\partial w_{m,n}^l} = o_m^{l-1};\ \frac{\partial i_n^l}{\partial b_n^l} = 1$$

[16] 由于这里讨论的神经网络是全连接的，因此i_n^l会通过o_n^l影响下一层的每一个神经元。正因为如此，在计算它的偏导数时，需要考虑所有的i_m^{l+1}。

仅供非商业用途或交流学习使用

12.3.3 算法步骤

反向传播算法的具体计算过程与 9.4.6 节讨论的最大期望算法非常类似，可以分为向前计算（forward pass）和向后计算（backwards pass）两类。

- 首先随机生成模型参数$w_{m,n}^l$和b_m^l。
- 根据现有的模型参数和训练数据，计算神经网络里每个神经元的i_n^l和o_n^l。
- 根据反向传播算法的数学公式（公式中的i_n^l和o_n^l是已知的），更新模型参数$w_{m,n}^l$和b_m^l（使用随机梯度下降法），并不断重复上一步和这一步，整个过程如图 12-13 所示。

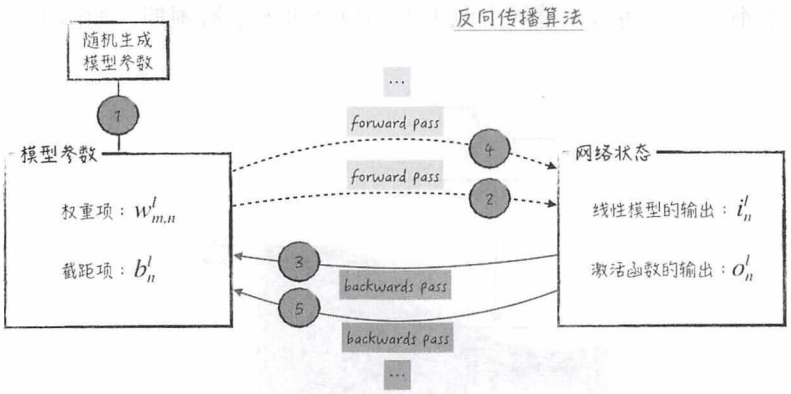


图 12-13

12.4 提高神经网络的学习效率

神经网络在训练过程中常常会碰到瓶颈期（可参考图 12-10 中的例子），在瓶颈期内，神经网络的模型效果几乎不再改进，仿佛已经达到它的极限一样。如果将神经网络想象成一个人，那么这个人的学习效率并不高，而本节的讨论重点就是如何提高神经网络的学习效率。

12.4.1 学习的原理

从数学上来看，神经网络在训练过程中，它的模型结构是固定的（也就是说数学公式是不变的），每次更新的只是模型参数的估计值。如果参数的估计值向“正确值”靠近得越快，那么神经网络也就学习得越快，因此，学习效率的核心是模型参数的更新策略。神经网络有两类模型参数，权重项$w_{m,n}^l$和截距项b_n^l。它们的更新策略和相关讨论是类似的，为了避免无谓的重复，下面只讨论神经网络的权重项$w_{m,n}^l$。

仅供非商业用途或交流学习使用

根据 12.3 节中的讨论，神经网络使用反向传播算法来训练模型，其中模型参数的更新原理是梯度下降法，具体的公式如下，其中γ是被称为学习速率是一个超参数，可以理解为一个不变的值。

$$(w_{m,n}^{l})_{j+1} - (w_{m,n}^{l})_{j} = -\gamma \frac{\partial L}{\partial w_{m,n}^{l}} \tag{12-21}$$

在公式（12-21）中，$(w_{m,n}^{l})_{j}$表示模型参数迭代j次后，$w_{m,n}^{l}$的估计值。根据这个公式，$-\gamma \frac{\partial L}{\partial w_{m,n}^{l}}$的符号决定了参数变化的方向（具体细节请参考第 6 章），而它的绝对值决定了变化的力度。可以将这个绝对值理解称为错误的惩罚，就跟人的学习过程类似，通常惩罚越大，人学得也就越快。但在神经网络的训练中，我们常常发现这个值很接近 0，导致模型参数在训练过程中几乎不变动，而这在整体上就表现为模型进入了瓶颈期，如图 12-14 所示。

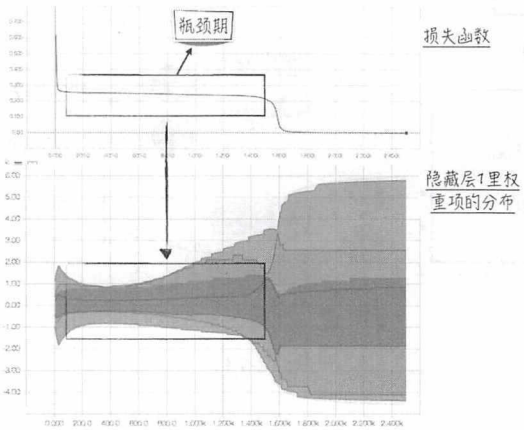


图 12-14

图 12-14 对应着图 12-10 中标记 3 的模型和数据，图中的上半部分是神经网络损失函数随迭代次数变化的曲线，而下半部分是隐藏层 1 里权重项的分布情况（具体的图示请参考 6.3.4 节）。可以看到，当参数变化幅度不大时，模型进入瓶颈期；而当参数大幅变化时，模型的损失快速降低。

根据上面的分析结果，解决神经网络学习效率低下的原理就很明显了，需要使公式（12-21）的右边，也就是$-\gamma \frac{\partial L}{\partial w_{m,n}^{l}}$，在模型训练的过程中明显不等于 0[17]。为了达到这个目的，比较直观的做法是，选择比较大的学习速率γ，这也是其他模型比较常用也比较有效的方法。但不幸的是，

[17] 这样的表述并不严谨。事实上，当参数估计值靠近或者等于“真实值”时，我们需要$-\gamma \frac{\partial L}{\partial w_{m,n}^{l}}$接近或者等于 0，这样才不会“错过真实值”。因此严谨的表述应该是，当参数估计值明显远离“真实值”时（也就是损失函数远离其极值点时），需要保证$-\gamma \frac{\partial L}{\partial w_{m,n}^{l}}$明显不等于 0。

仅供非商业用途或交流学习使用

这个方法对神经网络几乎没什么作用，具体的原因很复杂，我们将在之后的 12.4.4 节中讨论。于是只剩下一种选择，就是想办法使得偏导数$\frac{\partial L}{\partial w_{m,n}^{l}}$远离 0，而这正是下面将讨论的内容。

12.4.2 激活函数的改进

如图 12-15 所示，根据反向传播算法，激活函数的导数f'将影响偏导数$\frac{\partial L}{\partial w_{m,n}^{l}}$的值。因此，先讨论激活函数的改进。

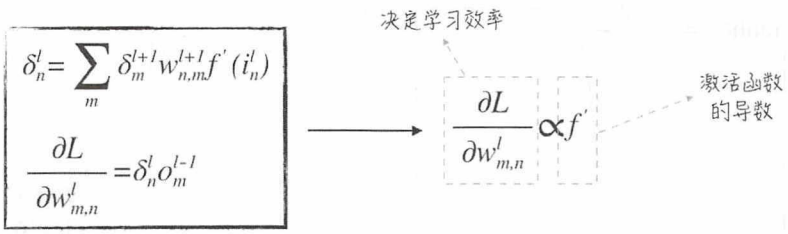


图 12-15

在之前的讨论中，神经网络的激活函数是 sigmoid 函数。这个函数虽然在理论上是一个很好的选择，但它有一个对工程实现不太友好的特点：当自变量的值远离 0 时，函数的导数值接近 0，如图 12-16 所示。这就导致使用 sigmoid 函数作为激活函数时，神经网络的学习效率通常很低。

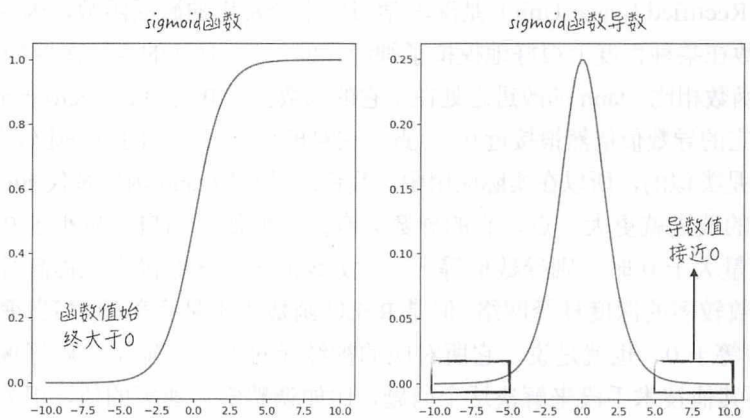


图 12-16

事实上，除了函数导数值接近 0 外，sigmoid 函数还有另一个不太友好的特点是，它的输出总是正数，并不以 0 为中心（not zero-centered）。这会导致在训练的过程中，损失函数

仅供非商业用途或交流学习使用

的梯度都变成正数或者都变成负数（具体的证明过程很复杂，在此就不进行展开介绍了），从而使模型的学习效率降低。

为了改进这两个不太友好的缺点，学术界提出了许多新的激活函数[18]。下面介绍其中应用最为广泛的两个。

- tanh（hyperbolic tangent）的函数曲线与 sigmoid 函数非常类似，可以认为是 sigmoid 函数向下平移了一段，使得函数值的中心等于 0，如图 12-17 所示。

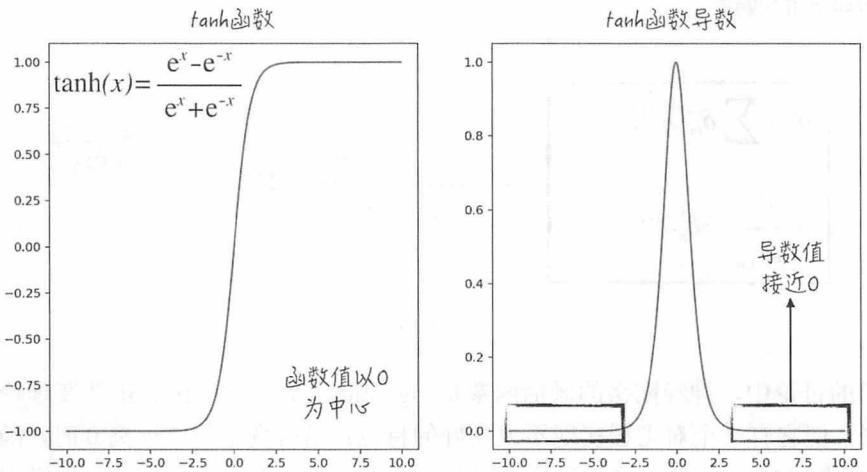


图 12-17

- ReLU（Rectified Linear Unit）是深度学习中十分常用的激活函数，因为从生物学的角度来看，这个函数在某种程度上很好地模拟了神经元的行为。具体的函数图像如图 12-18 所示。

与 sigmoid 函数相比，tanh 的改进之处在于它的函数值以 0 为中心（zero-centered），但当自变量远离 0 时，它的导数值依然很接近 0。因此，它对模型学习效率的改进比较有限。但因为这两个函数的图像是类似的，所以在实际应用中，我们总是会用 tanh 函数替代 sigmoid 函数。

ReLU 函数的改进就更大一点，它的导数只有两个取值，当自变量小于 0 时，则导数值等于 0；当自变量大于 0 时，则导数值等于 1。实践证明，这能极大地提高神经网络的学习效率，特别是层数较多的深度神经网络。但是 ReLU 函数也不是完美的，当自变量小于 0 时，则它的函数值恒等于 0。也就是说，它所对应的神经元对外没有输出，就好像“死”掉了一样，这时就需要其他技术手段来解决这个问题，比如调整学习速率的值，相关的内容超出本书的范围，在此就不做讨论了。

[18] 激活函数是神经网络研究中比较热门的领域，除了正文中讨论的这两个激活函数，常用的函数还有 leak ReLU、ELU（exponential linear units）以及 maxout，相关的细节请参考维基百科。

仅供非商业用途或交流学习使用

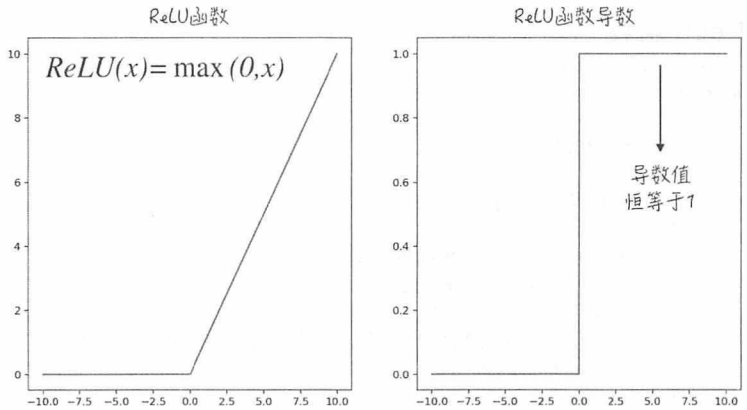


图 12-18

针对 12.2.3 节中的例子（使用图 12-9 中标记 4 的数据），如果使用 ReLU 函数作为激活函数搭建神经网络（神经网络的结构仍然是[4, 4, 2]），可以得到如图 12-19 所示的结果：相比于之前（使用 sigmoid 函数作为激活函数），神经网络的学习效率得到了很大的提高。

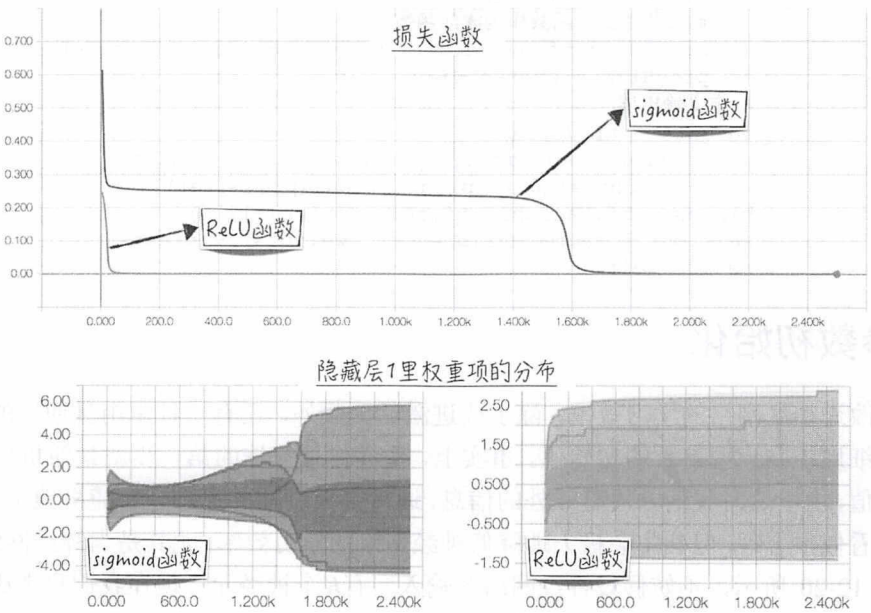


图 12-19

更换激活函数的代码实现非常简单，只需在定义神经网络隐藏层时，使用 tf.nn.relu 函数即可，如程序清单 12-5 中第 24 行代码所示。

仅供非商业用途或交流学习使用

程序清单 12-5 使用 ReLU 函数

```
class ANN(object):
    # 省略掉其他部分

    def defineANN(self):
        """
        定义神经网络的结构
        """
        # self.input 是训练数据里自变量
        prevSize = self.input.shape[1].value
        prevOut = self.input
        # self.size 是神经网络的结构，也就是每一层的神经元个数
        size = self.size
        layer = 1
        # 定义隐藏层
        for currentSize in size[:-1]:
            weights = tf.Variable(
                tf.truncated_normal([prevSize, currentSize],
                    stddev=1.0 / np.sqrt(float(prevSize))))
            # 记录隐藏层的模型参数
            tf.summary.histogram("hidden%s" % layer, weights)
            layer += 1
            biases = tf.Variable(tf.zeros([currentSize]))
            # 使用 ReLU 函数作为激活函数
            prevOut = tf.nn.relu(tf.matmul(prevOut, weights) + biases)
            prevSize = currentSize
        # 定义输出层
        weights = tf.Variable(
            tf.truncated_normal([prevSize, size[-1]],
                stddev=1.0 / np.sqrt(float(prevSize))))
        biases = tf.Variable(tf.zeros([size[-1]]))
        self.out = tf.matmul(prevOut, weights) + biases
        return self
```

12.4.3 参数初始化

为了改进神经网络的学习效率，除了改进激活函数外，还有一种常用且通用的方法，那就是更精细地控制模型参数的初始化。事实上，反向传播算法的第一步就是随机生成模型参数的初始值，由于这时我们并没有额外的信息，通常会根据标准正态分布随机地生成初始值。这个方法看似很合理，但其实会极大地降低神经网络的学习效率，尤其是当神经网络很大时。

如图 12-20 所示，不妨假设神经网络的输入层有K个神经元。现在我们只考虑隐藏层 1 里的权重项$w_{m,1}^{1}$。这些参数的初始值是相互独立的，且服从标准正态分布（记为$\mathcal{N}(0,1)$）。由于隐藏层 1 里的线性模型输出i_1^1是这些值的线性组合，那么它服从期望等于 0，方差等于K的正态分布（记为$\mathcal{N}(0,K)$），也就是说，i_1^1有很大的概率等于一个绝对值非常大的数，而后者会给模型训练带来困难。

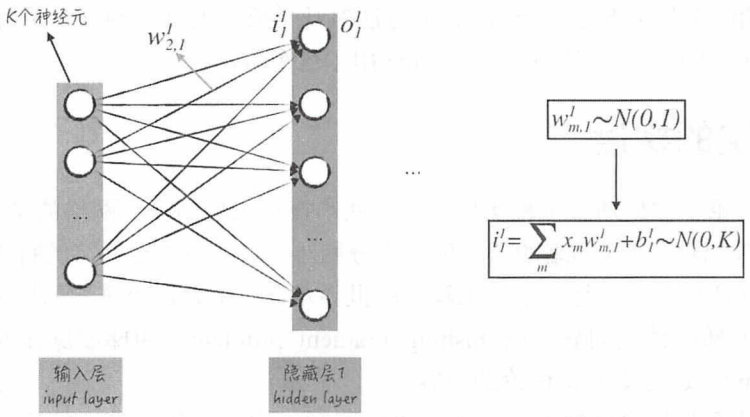


图 12-20

以 sigmoid 函数为例，当i_1^1绝对值很大时，导数值$f'(i_1^1)$会很接近 0，从而导致偏导数$\frac{\partial L}{\partial w_{m,1}^1}$的值很接近 0，影响模型的学习效率。即使改用效率更高的 ReLU 函数，也同样会引起问题。当i_1^1是很大的正数时，虽然这时函数的导数值等于 1，但相应的激活函数输出o_1^1是一个很大的正数，这会导致隐藏层 2 里梯度很大，从而使模型训练的效率不高，因为梯度过大会导致参数估计值错过“真实值”（具体细节请参考第 6.2 节）。

事实上，不管激活函数怎么选择，线性输出i_n^l过大都会导致问题。因为反向传播算法里既用到了激活函数的导数$f'(i_n^l)$，又用到了激活函数本身$o_n^l = f(i_n^l)$，所以这两个值没办法在i_n^l很大时，同时保证“合理”（远离 0 且绝对值不大），如图 12-21 所示。

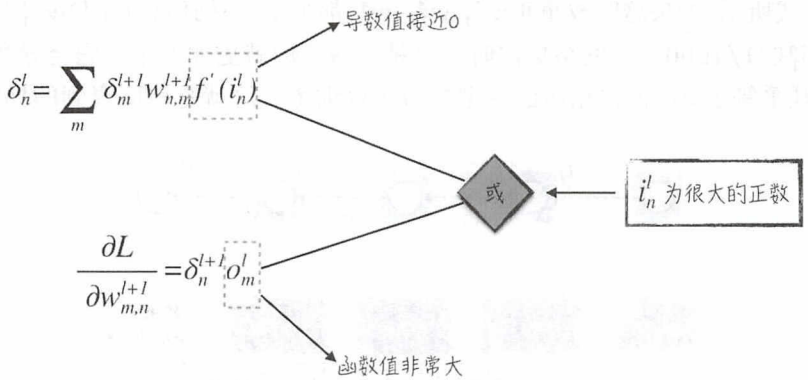


图 12-21

由于这个问题是由正态分布的线性组合引起的，那么解决方法就很明显了，在参数初始化时，把正态分布的标准差调整为$1/\sqrt{M}$，其中M为上一层里的神经元个数。这样设定后，

线性输出i_n^l将始终服从标准正态分布。这个方法对几乎所有的神经网络都是有用的，事实上，我们在程序清单 12-1 和程序清单 12-5 中都使用了这个方法。

12.4.4 不稳定的梯度

上面我们讨论了如何通过工程实现上的一些小技巧来提高神经网络的学习效率，但即使应用了这些技巧，深度神经网络的训练依然十分困难。这是由神经网络的特殊模型结构以及估算参数的方法所引起的，是一个尚未解决的世界难题。引起这个难题的原因有两个，在学术上分别被称为梯度消失问题 （vanishing gradient problem）和梯度爆炸问题（exploding gradient problem），这是本节将讨论的内容。

为了理解什么是消失的梯度，让我们再次回顾一下反向传播算法的计算步骤，如公式（12-22）所示，其中$\delta_n^l = \frac{\partial L}{\partial i_n^l}$; $o_n^l = f(i_n^l)$ 。

$$\begin{aligned} \delta_n^T &= \frac{\partial L}{\partial i_n^T} = \frac{\partial L}{\partial o_n^T} \\ \delta_n^l &= \sum_m \delta_m^{l+1} w_{n,m}^{l+1} f'(i_n^l) \\ \frac{\partial L}{\partial b_n^l} &= \delta_n^l; \frac{\partial L}{\partial w_{m,n}^l} = \delta_n^l o_m^{l-1} \end{aligned} \tag{12-22}$$

在上面的公式中，$\delta_n^l = \sum_m \delta_m^{l+1} w_{n,m}^{l+1} f'(i_n^l)$负责将模型错误向后传递，是最重要的一步计算。这步计算将涉及第$l+1$层里的所有权重项，并一直向后延伸到神经网络的输出层。这种叠加效应很容易导致δ_n^l接近 0，从而导致第l层里的“梯度消失”（梯度接近 0）。

为了更直观地理解这个问题，让我们来看一个很简单的神经网络，如图 12-22 所示。这个神经网络一共有 4 层，但每一层里只有一个神经元。这个神经网络中，δ_1^1与δ_1^4的关系如图 12-22 中的公式所示。如果这时权重项$w_{1,1}^2, w_{1,1}^3, w_{1,1}^4$都很小，不妨假设$f'(i_1^l)w_{1,1}^{l+1}$都等于 0.1，则$\delta_1^1$约等于$\delta_1^4$的$1/1000$，当网络更深时，这种衰减效应就更加严重。它会导致网络前面几层的梯度都几乎等于 0，使得相应的模型参数难以训练。这就是前面提到的梯度消失问题。

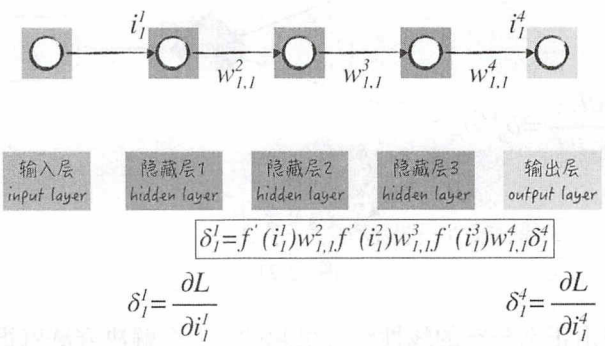


图 12-22

书籍是人类进步的阶梯

梯度爆炸问题与梯度消失问题十分类似。事实上，如果图 12-22 中的权重项$w_{1,1}^2, w_{1,1}^3, w_{1,1}^4$都很大，不妨假设$f'(i_1^l)w_{1,1}^{l+1}$都大于 10，则$\delta_1^1$约等于$\delta_1^4$的 1000 倍，这会导致网络前面几层的梯度都是绝对值非常大的数，也会使得相应的模型参数难以训练。

从上面的分析可以看到，梯度消失和梯度爆炸其实是同一个问题的两个面。这个问题产生的根本原因是网络后面几层的模型参数会影响网络前面几层参数的梯度。就如图 12-21 所示的那样，在网络层数较多或者网络较大时，参数与参数梯度（损失函数对参数的偏导数）这两个值无法同时保持“合理”。换句话说，神经网络的学习效率从根本上是难以提高的，因此也没办法在实际应用中搭建和使用复杂的神经网络[19]。

如果仔细分析梯度下降法的公式会发现，参数变动的速度其实是$-\gamma\frac{\partial L}{\partial w_{m,n}^l}$。除了参数的梯度外，这个值还会受到学习速率$\gamma$的影响。如果对网络中的不同层使用不同的学习速率，其实可以在一定程度上缓解各层梯度差别很大的问题。当然这样的做法会带来一系列其他的问题，相关的细节超出了本书的范围（这些问题是当前神经网络的研究热点，但没有一个广泛接受的解决方案），在此就不做讨论了。

12.5 本章小结

本章讨论的内容是神经网络，这也是目前数据科学领域最热门的话题。与其他模型相比，神经网络的建模思路是非常独特的。它并不是从人的主观出发，通过对现实的假设来搭建模型，而是单纯地通过模仿人的大脑来搭建模型。

具体来说，本章首先讨论了如何利用模型来仿造大脑的计算单位神经元。在模型中，神经元被分为两部分，第一部分是一个线性模型，它将接收到的信号进行线性加总并传给神经元的第二部分；第二部分是一个非线性的激活函数，它负责对初始信号进行变换形成最终的模型输出，是神经元中最核心的部分。从数学上来看，神经元中最经典的激活函数是 sigmoid 函数，因为这个函数模拟了两种相反效应的相互竞争，具有很强的生物学基础。但由于它的函数特性，使用 sigmoid 函数会使得模型训练变得特别困难，因此在它的基础上有很多改进的激活函数，其中最常用的是 ReLU 函数。

有了神经元模型后，接下来就是模仿大脑的结构，将多个神经元联结起来组成神经网络。本章讨论了最简单和直观的联结方式：将神经元分层，不同层的神经元首尾相接形成一个没有环的网络。在这种结构中，神经网络里的神经元被分为 3 个层级：输入层、隐藏层和输出

[19] 根据生物学的研究，人脑中有 860 亿个神经元。而到目前为此，人类仿造出的神经网络最多也只有几百万个神经元，因为当前的技术手段还没有办法处理更大、更复杂的神经网络。另外，相比于人脑的复杂结构，人类搭建的神经网络显得过于幼稚了，这也是为什么不少人认为我们仍处在人工智能的“石器时代”。

书籍是人类进步的阶梯

层。这 3 层中的神经元在结构上各不相同，其中只有隐藏层里的神经元才是完整的（同时具有线性模型和激活函数）。从模型效果来说，神经系统具有非常广泛的适用范围，也就是说，同样结构的神经网络可以处理不同分布类型的数据。当然到目前为止，与人相比，神经网络的适用范围还是很窄的，这也是现在人工智能被称为弱人工智能（artificial narrow intelligence）的原因。

讨论完神经网络的结构后，我们开始讨论神经网络的工程实现，这包括训练神经网络的反向传播算法以及提高神经网络学习效率的技巧。这些是数据科学这门学科里非常前沿的内容，很多问题都还没有普遍接受的解决方案，因此本章只介绍了其中的基础知识以及经典难题。

限于篇幅，本章只讨论了神经网络最经典的内容，想要了解更多的读者可参考其他书籍，比如 Tariq Rashid 编写的 *Make Your Own Neural Network* 以及 Michael Nielsen 编写的 *Neural Networks and Deep Learning*。

从结构上来看，本章讨论的神经网络有点太过简单了，它与人脑的真实结构还差得很远。我们可以在它的基础上，将其改造成更像大脑的结构。这就是深度学习所研究的内容，也是第 13 章讨论的重点。

书籍是人类进步的阶梯

第13章

深度学习：继续探索

When you hear the term deep learning, just think of a large deep neural net. Deep refers to the number of layers typically and so this kind of the popular term that's been adopted in the press.

（当你听到“深度学习”这个词时，把它想象成一个大型深度神经网络。在“深度学习”这个很流行的词里，“深度”二字表示神经网络的层数很多。）

——*Jeff Dean*

13.1 利用神经网络识别数字

13.2 卷积神经网络

13.3 其他深度学习模型

13.4 本章小结

书籍是人类进步的阶梯

在第 12 章中，我们讨论了经典的全连接神经网络。由于数学以及工程上的限制，这种结构的神经网络在实际应用中只能搭建和使用层数较少的浅层网络。但在生物学上，有很多证据都表明增加网络的层数能有效地提高模型的效果，因此学术界想尽办法要搭建层数更多、结构更为复杂的深度神经网络，这就是所谓的深度学习。

深度学习包含的模型非常多，但大多数模型还处于实验室阶段，很不成熟，而且到目前为止，即使是层数较少、结构较为简单的浅层神经网络，人类都还没办法理解它们，更谈不上复杂的深度学习模型了。根据数据科学的实践经验，如果一个模型难以解释，那么它很难被大规模使用，这可能是因为人类总是对不受控制的未知事物充满恐惧。因此，除了一些特定的应用场景，比如图片分类、语言识别等，要在实际的生产中大规模使用深度学习模型，估计还有很长的一段路要走。

针对深度学习，本章首先讨论其中应用很广泛且模型成熟度较高的卷积神经网络，它常被用于图片识别领域。接着我们将简单介绍另一类比较热门的深度学习模型——递归神经网络。这个网络引入了所谓的记忆机制，在自然语言处理以及时间序列分析领域有十分惊艳的表现。

从模型的分类上来看，卷积神经网络和递归神经网络都是监督式学习，而本章的最后将介绍深度学习在非监督式学习中的探索。虽然从搭建模型的角度来讲，非监督式深度学习的研究更加困难，因为并没有客观存在的标准来评估模型效果。但从现实的角度来讲，人在生活中的学习模式往往是非监督式：生活并不告诉我们答案，但人却能从生活中学到很多东西。如果深度学习“学会”了人的这种能力，那么机器就完全可以独立且自主地进行学习和进化。这也是为什么不少人都认定非监督式深度学习是人工智能实现突破的希望。当然现在还很难想象那种状态下的人工智能会是什么样子，但可以肯定的是，它将更加不受人的控制，也没人知道它在学习什么，或者说它在思考和酝酿着什么。

13.1 利用神经网络识别数字

在上一章中，我们展示了如何使用神经网络来解决分类问题。不过为了更加直观地展示神经网络的主要特点，只讨论使用了人造的二维数据。本节我们将讨论一个真实的例子：如何使用神经网络来识别数字图片。

13.1.1 搭建模型

我们将搭建神经网络对 MNIST 图片集进行识别。MNIST 图片集是 0～9 的手写数字图片，如图 13-1 所示，其中包含 60 000 张训练图片和 10 000 张测试图片。

书籍是人类进步的阶梯

图 13-1[1]

与日常生活中接触到的图片不同，MNIST 里的图片已经转换为了数字形式[2]。具体来说，MNIST 里的图片在横向有 28 个像素点，纵向也有 28 个像素点。由于图片是黑白的，图片中的每一个像素点由一个正数表示，这个数字表示该点的黑白程度。因此在 MNIST 数据集中，图片被存储为$28 \times 28 = 784$个数字（这其实就是计算机眼中的图片），图片与数字之间的关系如图 13-2 所示。值得注意的是，为了展示方便，我们将原本28×28数字图片压缩到8×8。另外，虽然图 13-2 中用矩阵的形式来展示数据，但它其实是一个 64 维的行向量。

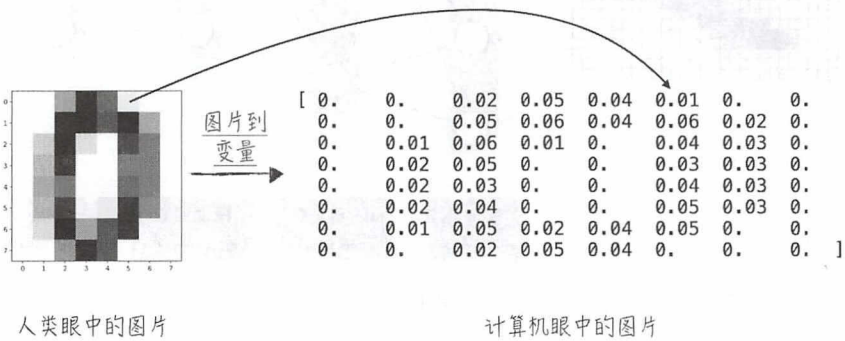


图 13-2

[1] MNIST 图片集是神经网络研究领域最常用的数据之一，常被用来评估不同模型的效果。这个图片集存储格式比较特殊，需要使用专门的程序将其解析。这个数据集的下载地址以及解析程序请参考随书配套的代码/ch13-deep_learning/util.py。

[2] 使用模型处理图片时，需要将后者转换为数字形式，因此对于普通图片，通常需要使用其他工具，比如第 10 章中用到的第三方库 pillow。当然如果图片是以数字形式存在的，那么并不能像普通图片那样双击打开它，而是需要用程序将其可视化。

书籍是人类进步的阶梯

从机器学习的角度来看，表示一张图片的 784 个数字其实对应着 784 个变量，每个数字对应着相应变量的取值。因此，把图片转换为数字形式实际上就是将图片转换为我们所熟悉的建模数据。

介绍完建模数据，接下来将讨论如何搭建神经网络。对于输入层，由于数据有 784 个自变量，因此这一层有 784 个神经元，每个神经元依次对应着一个自变量。对于输出层，由于数据分为 10 类，因此这一层有 10 个神经元，每一个神经元对应着一个类别。与之前的分类问题类似，将输出层的结果传给 softmax 函数便可以得到每个类别的预测概率。对于隐藏层的选择就比较自由了，由于深度的神经网络很难训练，因此只搭建两层隐藏层，它们的神经元个数分别为 30 和 20。整个神经网络的结构如图 13-3 所示。

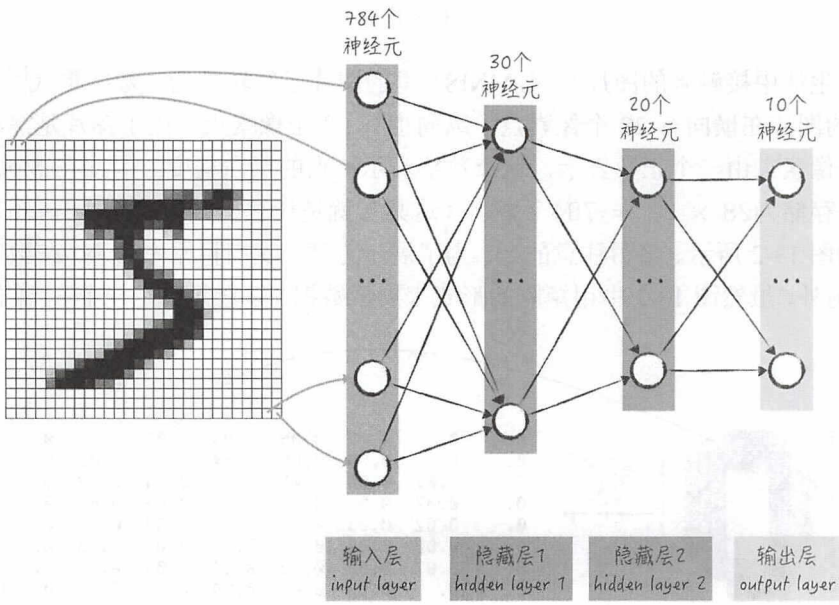


图 13-3

13.1.2 防止过拟合之惩罚项

在第 12 章中，我们并没有将数据集划分为训练集和测试集，也就是没有考虑模型过拟合（overfitting）的问题，但这其实是神经网络中十分常见和严重的问题，因为神经网络是极其复杂的模型，极易发生过拟合。本节将讨论如何在神经网络里防止过拟合。

与其他模型类似，可以通过在损失函数里增加惩罚项来防止神经网络过拟合。事实上，在第 4.3.3 节中，我们讨论了如何在线性模型里使用 L2 惩罚项以及 L1 惩罚项，这里的方法与之类似。

书籍是人类进步的阶梯

- L2 惩罚项。不妨假设神经网络原本的损失函数为L，则如公式（13-1）所示，定义新的损失函数$\bar{L}$。其中λ为惩罚项的权重，是模型的超参数，而W表示神经网络里的所有权重项，不妨记为$\boldsymbol{W} = (w_1, \ldots, w_N)$，则$\|\boldsymbol{W}\|^2 = \sum_i w_i^2$。

$$\bar{L} = L + 0.5\lambda\|\boldsymbol{W}\|^2 \tag{13-1}$$

- L1 惩罚项。这个惩罚项与 L2 惩罚项非常类似，如果使用相同的记号，则可以得到如公式（13-2）所示的新损失函数，其中$\|\boldsymbol{W}\|_1 = \sum_i |w_i|$。

$$\bar{L} = L + \lambda\|\boldsymbol{W}\|_1 \tag{13-2}$$

从直观上来讲，上面两个惩罚项降低过拟合问题的原理是让模型参数更加接近 0，这与线性回归模型里的情况是类似的。增加惩罚项可以避免训练数据的偏差使得模型过于依赖某些变量（对应的模型参数绝对值很大）的现象。

这两种惩罚项的差别在于，L1 惩罚项更倾向于得到“稀疏”的模型参数，也就是说只有少量的模型参数不等于 0。从宏观上来看，这时神经网络其实只使用了部分变量，因此抗噪能力比较强（噪声变量的权重通常被估计为 0）。而 L2 惩罚项则倾向于得到接近 0 但非 0 的参数估计。从宏观上来看，这时模型会用到几乎所有的变量，但每个变量对最终结果的影响都比较有限。

这两种惩罚项各有优缺点，需要根据具体的应用场景进行选取。也可以将两者进行综合得到所谓的 elastic net regularization，它的具体公式如下。其中，lw为模型的超参数，取值范围是 0～1，表示 L1 惩罚项的权重，可以通过网点搜寻（grid search）来得到这个变量的最佳取值。

$$\bar{L} = L + \lambda[lw\|\boldsymbol{W}\|_1 + (1 - lw)\|\boldsymbol{W}\|^2] \tag{13-3}$$

13.1.3 防止过拟合之 dropout

惩罚项是解决过拟合问题比较通用的方案，但对于神经网络这种特别复杂的模型，仅使用这种方法通常是不够的。为了更好地防止神经网络发生过拟合，学术界引入了一种很特殊的方法——dropout[3]。这个方法通常与惩罚项一起使用，配合着解决过拟合问题。概括来讲，dropout 的思路十分独特：在训练模型的每一步中，随机地暂时剔除掉一些神经元[4]。当然，在使用神经网络对未知数据做预测时，会使用完整的网络，并不丢弃任何神经元。

为了便于讨论 dropout 算法，不妨假设搭建的神经网络为 3 层，具体结构为[3, 4, 2]，如

---

[3] 对神经网络使用 dropout 方法除了能防止过拟合外，还能在某种程度上实现神经网络的 bagging training（类似于第 8 章中讨论的随机森林，它其实就是决策树模型的 bagging training）。因此 dropout 能有效地提升神经网络的模型效果。具体细节请参考 Ian Goodfellow 等人编写的 *Deep Learning*。

[4] 能被 dropout 剔除的是隐藏层和输入层的神经元（通常只剔除隐藏层里的神经元）。输出层的神经元并不能被剔除，否则相当于随机排除掉一些预测结果，这并不能提升模型的效果。

书籍是人类进步的阶梯

图 13-4 所示。在正常情况下，使用反向传播算法训练这个神经网络时，它将始终保持这个结构。但是如果使用 dropout，那么神经网络的一次训练（随机梯度下降法的一次迭代）将变成如下的步骤。

- 在训练开始之前，先从神经网络中随机剔除一些神经元，使之变成一个新的神经网络，比如在图 13-4 中，在某一次的训练中，原本的神经网络被变成了[2, 2, 2]的结构。
- 然后根据这个新的网络结构，使用反向传播算法更新神经元的参数，而剔除掉的神经元在这一步里并不更新它的参数。
- 每次训练结束之后，将神经网络恢复到最初的结构，并进入下一次训练。

整个过程如图 13-5 所示。

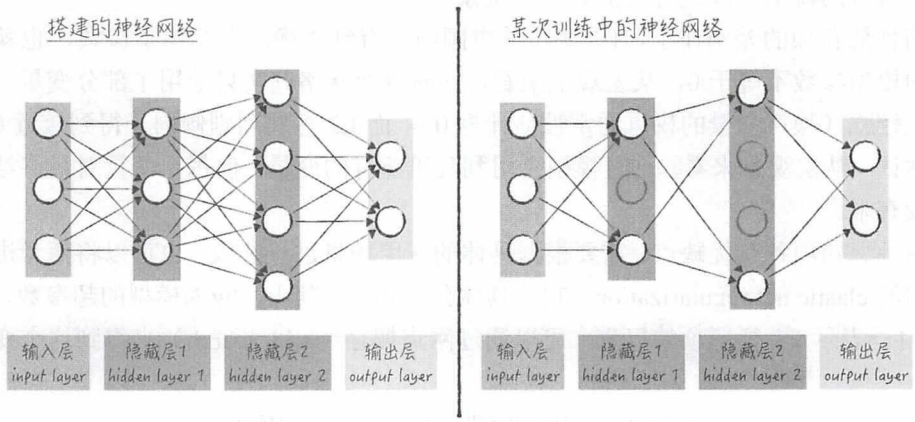


图 13-4

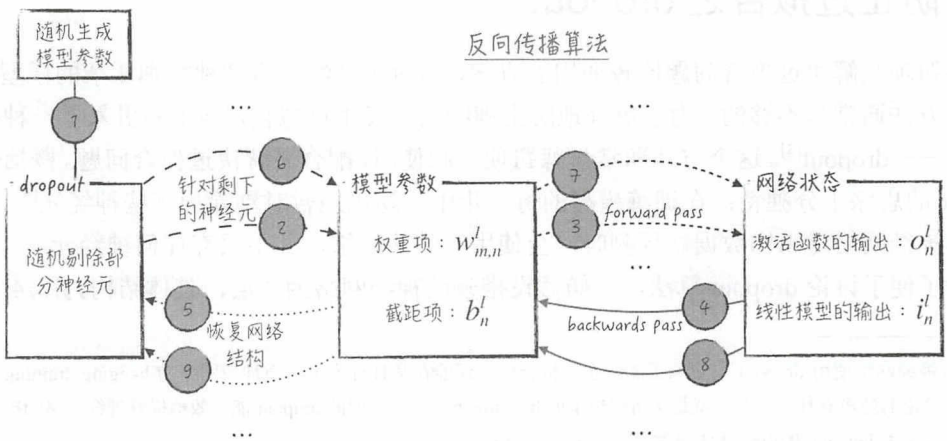


图 13-5

书籍是人类进步的阶梯

正如前面提到的，在使用神经网络对未知数据做预测时，我们并不会丢弃网络中的任何神经元，这会造成神经元在训练中的“作用”和使用时的“作用”并不一样。为了说明这个问题，不妨假设在 dropout 过程中，每个神经元被保留的概率等于p。那么针对第l层的第n个神经元，在使用模型时，它相应的正常输出为o_n^l（它会一直输出这个值）。但在训练时由于 dropout，它有时输出o_n^l，有时输出 0（当这个神经元被剔除时），平均输出为po_n^l。这种现象会造成整个神经网络在训练时的表现和使用时相差很大。为了消除这种差异，在使用 dropout 训练神经网络时，需要相应地将神经元输出扩大到原来的$1/p$倍。

13.1.4 代码实现[5]

本节将讨论如何用代码来实现上面讨论的模型。由于第 12 章中已经讨论了如何实现最基本的神经网络，因此，下面的讨论重点是如何实现防止神经网络过拟合的各种方案。

第一步是定义神经网络的结构，并在定义的过程中加入 dropout 方法，如程序清单 13-1 所示。

（1）与第 12 章类似，我们使用类（class）来实现神经网络。这个类里的“self.input”对应着训练数据里的自变量，“self.size”是神经网络的结构。

（2）为了在神经网络的损失函数中加入惩罚项，定义“self.W”变量，它将记录神经网络中的所有权重项，如第 14、21、39 行代码所示。

（3）正如 12.4.2 节中讨论的，这个神经网络将使用 ReLU 函数作为激活函数，因为这将提高神经网络的学习效率，具体的实现如第 28 行代码所示。

（4）与惩罚项不同，dropout 需要定义在神经网络的结构上，如第 30 行代码所示。其中，“neuralOut”是当前隐藏层的“正常输出”；而“self.keepProb”表示每个神经元在训练中被保留的概率（它的类型是 tf.placeholder，因此可以在使用神经网络的过程中更改这个变量的值）。这个变量的取值范围是$(0,1]$，当它等于 1 时，完整的神经网络将被使用（对未知数据做预测时，需要完整的神经网络）。值得注意的是，在 TensorFlow 中，tf.nn.dropout 已经自动地将神经元输出扩大到“正常输出”的$1/self.keepProb$倍。

经过 dropout 处理后，神经元的输出为“prevOut”，它将参与下一层神经元的定义。整个隐藏层的结构如图 13-6 所示。

[5] 完整的实现请参考随书配套的代码/ch13-deep_learning/mlp.py。

书籍是人类进步的阶梯

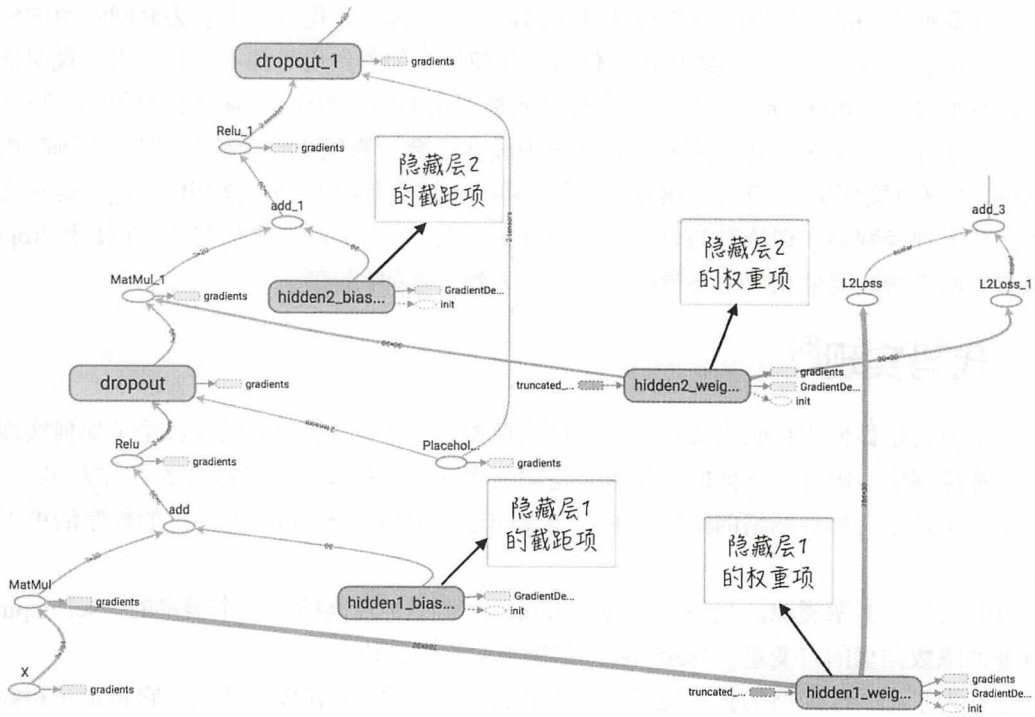


图 13-6

程序清单 13-1 定义神经网络的结构

```
class ANN(object):
    # 省略掉其他部分

    def defineANN(self):
        """
        定义神经网络的结构
        """
        # self.input 是训练数据里自变量
        prevSize = self.input.shape[1].value
        prevOut = self.input
        # self.size 是神经网络的结构，也就是每一层的神经元个数
        size = self.size
        layer = 1
        self.W = []
        # 定义隐藏层
        for currentSize in size[:-1]:
            weights = tf.Variable(tf.truncated_normal(
                [prevSize, currentSize], stddev=1.0/np.sqrt(float(prevSize))),
                name="hidden%s_weights" % layer)
            # 将模型中的权重项记录下来，用于之后的惩罚项
            self.W.append(weights)
```

书籍是人类进步的阶梯

书籍是人类进步的阶梯


```
22 |             # 记录隐藏层的模型参数
23 |             tf.summary.histogram("hidden%s" % layer, weights)
24 |             biases = tf.Variable(tf.zeros([currentSize]),
25 |                 name="hidden%s_biases" % layer)
26 |             layer += 1
27 |             # 定义这一层神经元的输出
28 |             neuralOut = tf.nn.relu(tf.matmul(prevOut, weights) + biases)
29 |             # 对隐藏层里的神经元使用 dropout
30 |             prevOut = tf.nn.dropout(neuralOut, self.keepProb)
31 |             prevSize = currentSize
32 |         # 定义输出层
33 |         weights = tf.Variable(tf.truncated_normal(
34 |             [prevSize, size[-1]], stddev=1.0 / np.sqrt(float(prevSize))),
35 |             name="output_weights")
36 |         biases = tf.Variable(tf.zeros([size[-1]]), name="output_biases")
37 |         self.out = tf.matmul(prevOut, weights) + biases
38 |         # 将模型中的权重项记录下来，用于之后的惩罚项
39 |         self.W.append(weights)
40 |         return self
```

第二步是定义模型的损失函数，并在其中加入惩罚项，如程序清单 13-2 所示。

（1）模型预测损失的定义如第 9～11 行代码所示，其中，“self.label”对应着训练数据里的标签变量，由于预测的类别为 10 个，因此它是一个 10 维的向量（对标签变量使用 One-Hot Encoding 得到的结果）。

（2）这个例子将使用 L2 惩罚项，它相应的实现如第 13、14 行代码所示，如果想使用 L1 惩罚项，则相应的实现为 tf.contrib.layers.l1_regularizer。模型整体的损失等于预测的损失加上惩罚项，如第 16、17 行代码所示，其中，“self.lambda_”表示惩罚项的权重，对于神经网络，这个值通常的范围是[0.001, 0.1]。

（3）另外需要特别注意的是，如果使用随机梯度下降来训练模型，则模型在每次迭代时，只会用到部分数据。比如在随机梯度下降法中，每次迭代只用到1/100的训练数据，那么，第 11 行代码定义的“loss”只是模型“真实”预测损失的1/100，而 L2 惩罚项却不受影响。为了消除随机梯度下降法带来的干扰，需要相应地将惩罚项权重“self.lambda_”变为原定值的1/100。

程序清单 13-2 定义损失函数

```
 1 | class ANN(object):
 2 |     # 省略掉其他部分
 3 |
 4 |     def defineLoss(self):
 5 |         """
 6 |         定义神经网络的损失函数
 7 |         """
 8 |         # 定义单点损失，self.label 是训练数据里的标签变量
 9 |         loss = tf.nn.softmax_cross_entropy_with_logits(
10 |             labels=self.label, logits=self.out)
11 |         loss = tf.reduce_mean(loss)
```

书籍是人类进步的阶梯

```
12 |         # L2 惩罚项
13 |         _norm = map(lambda x: tf.nn.l2_loss(x), self.W)
14 |         regularization = reduce(lambda a, b: a + b, _norm)
15 |         # 定义整体损失
16 |         self.loss = tf.reduce_mean(loss + self.lambda_ * regularization,
17 |             name="loss")
18 |         # 记录训练的细节
19 |         tf.summary.scalar("loss", self.loss)
20 |         return self
```

第三步是定义如何使用神经网络对未知数据做预测，如程序清单 13-3 所示。由于这个例子里的神经网络使用了 dropout，因此在对未知数据做预测时，除了需要输入数据的自变量外，还需要将“self.keepProb”设置为 1，如第 10 行代码所示。

程序清单 13-3 对未知数据做预测

```
1 | class ANN(object):
2 |     # 省略掉其他部分
3 |
4 |     def predict_proba(self, X):
5 |         """
6 |         使用神经网络对未知数据进行预测
7 |         """
8 |         sess = self.sess
9 |         pred = tf.nn.softmax(logits=self.out, name="pred")
10 |        prob = sess.run(pred, feed_dict={self.input: X, self.keepProb: 1.0})
11 |        return prob
```

最后是训练并评估模型，如程序清单 13-4 所示。在随机梯度下降法中，学习速率是一个很重要的模型超参数（具体细节请参考第 6.2 节），它将决定参数每次更新的幅度。在之前的实现中，我们大多使用固定的学习速率，如果在训练初期使用较大的学习速率，在训练后期使用较小的学习速率，那么模型训练的速度将更快。实现这种可变学习速率的方法有很多，比较常用的是指数衰减学习速率，如第 10～13 行代码所示。这段代码表示，在训练的最开始，学习速率等于“startLearningRate”，每训练 1000 次，学习数据将变为之前的 96%。

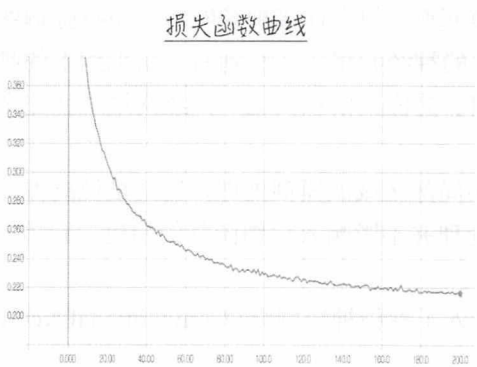
程序清单 13-4 训练并评估模型

```
1 | class ANN(object):
2 |     # 省略掉其他部分
3 |
4 |     def SGD(self, X, Y, startLearningRate, miniBatchFraction, epoch,
keepProb):
5 |         """
6 |         使用随机梯度下降法训练模型
7 |         """
8 |         summary = tf.summary.merge_all()
9 |         trainStep = tf.Variable(0)
10 |        learningRate = tf.train.exponential_decay(startLearningRate,
11 |            trainStep, 1000, 0.96, staircase=True)
12 |        method = tf.train.GradientDescentOptimizer(learningRate)
```

书籍是人类进步的阶梯

```
13 |         optimizer= method.minimize(self.loss, global_step=trainStep)
14 |         batchSize = int(X.shape[0] * miniBatchFraction)
15 |         batchNum = int(np.ceil(1 / miniBatchFraction))
16 |         sess = tf.Session()
17 |         self.sess = sess
18 |         init = tf.global_variables_initializer()
19 |         sess.run(init)
20 |         summary_writer = tf.summary.FileWriter(self.logPath, graph=tf.get_
                                 default_graph())
21 |         step = 0
22 |         while (step < epoch):
23 |             for i in range(batchNum):
24 |                 batchX = X[i * batchSize: (i + 1) * batchSize]
25 |                 batchY = Y[i * batchSize: (i + 1) * batchSize]
26 |                 sess.run([optimizer], feed_dict={self.input: batchX,
27 |                     self.label: batchY, self.keepProb: keepProb})
28 |             step += 1
29 |             # 评估模型效果，并将日志写入文件
30 |             self.evaluation(step)
31 |             summary_str = sess.run(summary, feed_dict={self.input: X,
32 |                 self.label: Y, self.keepProb: 1.0})
33 |             summary_writer.add_summary(summary_str, step)
34 |             summary_writer.flush()
35 |         return self
```

与其他模型相比，神经网络的训练十分困难，因此很难在模型训练开始前估计所需的迭代次数。相应的解决办法是在训练的过程中，监控模型的训练效果。如第 30 行代码所示，在每个训练轮次（epoch）完成之后，评估一下当前模型的效果，具体的结果如图 13-7 所示。在这个例子中，惩罚项的权重为 0.001，即“self.lambda_=0.001”，而 dropout 过程中，神经元的保留概率为 0.7，即“self.keepProb=0.7”。



模型效果

epoch 1
训练集的准确率 0.807
验证集的准确率 0.808
测试集的准确率 0.817
……
epoch 100
训练集的准确率 0.961
验证集的准确率 0.949
测试集的准确率 0.952
……
epoch 200
训练集的准确率 0.967
验证集的准确率 0.951
测试集的准确率 0.956

图 13-7

书籍是人类进步的阶梯

如果对于同样的神经网络，不使用防止过拟合的手段，也就是说“self.lambda_=0”以及“self.keepProb=1”。模型效果如图 13-8 所示，可以看到模型在训练集上的效果更好，当训练轮次等于 200 时，模型的准确率达到了 0.998，但模型在验证集和测试集都比之前差一点，准确率分别为 0.941 和 0.946。这说明模型发生了过拟合的问题，如果将神经网络设计得更复杂一点（层数更多或者每层的神经元更多），这样的问题将更加严重。

模型效果

epoch 1	epoch 100	epoch 200
训练集的准确率 0.840	训练集的准确率 0.988	训练集的准确率 0.998
验证集的准确率 0.841	验证集的准确率 0.941	验证集的准确率 0.941
测试集的准确率 0.848	测试集的准确率 0.946	测试集的准确率 0.944
……	……	

图 13-8

13.2 卷积神经网络[6]

全连接神经网络虽然分类效果不错（模型准确率达到了 95.6%），但与人相比，它还有两个明显的缺陷。

一方面，人在识别图片时，不仅关注每个像素点的颜色，还关心像素点之间的空间位置关系。也就是说，人往往会特别关注图片的局部信息，即相邻像素点所包含的信息，因为两个像素点离得越近，它们之间的相关性也就越强。但全连接神经网络却没有强调图片像素点的位置关系。如图 13-9 所示，隐藏层 1 中的神经元将按从左到右、从上到下的顺序依次连接图片的每一个像素点，那么图片中上下相邻的位置信息在这个神经网络里就完全没有被捕捉到。

另一方面，人眼在识别图片时，还会在局部区域做模糊处理，也就是说离得非常近的像素点会被认为是相同的颜色，这样的模糊处理并不影响人识别图片的精度。显然上面讨论的神经网络也没有这个特性。

为了改进上述的这两个缺陷，学术界引入了卷积神经网络（Convolutional Neural Network，CNN），下面将讨论这个模型的细节。

[6] 本节使用的插图大多参考自 Michael Nielsen 编写的 *Neural Networks and Deep Learning*。

书籍是人类进步的阶梯

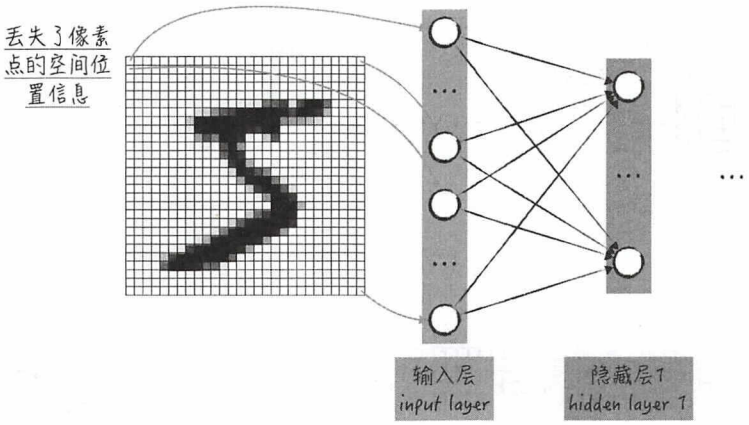


图 13-9

13.2.1 模型结构之卷积层

为了能捕捉到像素点的位置信息，卷积神经网络首先引入了所谓的 local receptive fields（不妨将其翻译为局部感受野[7]）。local receptive fields 指的是输入层到隐藏层 1 的一种特殊处理。它不再是全连接的，而是根据像素点的二维空间位置，将局部输入传给下一层的神经元，如图 13-10 所示。

与之前的神经网络一样，输入层一共有$28 \times 28 = 784$个神经元。为了直观展现，我们将其表示成矩阵的形状，其中的每一个小方块表示一个神经元，对应着相应位置的像素点。输入层的神经元按5×5的正方形区域（local receptive fields 的大小）依次与隐藏层 1 里的神经元相连，比如对于隐藏层 1 里的神经元 A，输入层里只有位于左上角的 25 个神经元（在图中用黑色标记）与之相连。代表 local receptive fields 的正方形区域每一次只移动一步（也就是说正方形区域在移动过程中有所重叠，这也是卷积神经网络的名称来源[8]），因此，隐藏层 1 里有24×24个神经元[9]。

---

[7] 感受野（receptive field）是一个生物学概念。根据维基百科上的介绍，一个感觉神经元的感受野是指能够引起该神经元反应的区域。

[8] 对数学比较熟悉的读者可能知道，卷积是一个很深刻的数学概念。假设f, g是两个可积函数，则它们的卷积定义为$h(x) = \int f(t)\, g(x - t)\mathrm{d}t$。如果$g(x)$的函数图像是一个正方形，而$f$的函数图像就是图片，那么两者卷积的计算过程就近似于图 13-10 中 local receptive fields 的移动过程。

[9] 事实上，local receptive fields 的移动步伐和边界处理有不同的算法。在不同的算法下，隐藏层 1 里的神经元个数有所差异，但这并不影响模型的整体结构，相关的细节将在代码实现部分讨论。

书籍是人类进步的阶梯

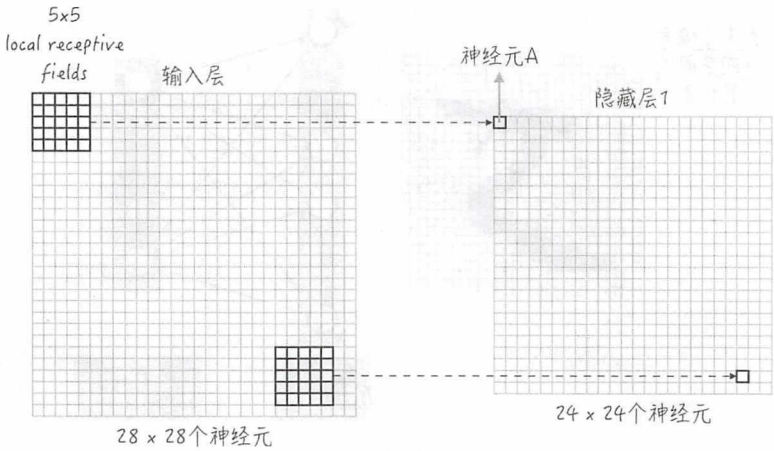


图 13-10

在之前讨论的神经网络中，隐藏层里的神经元输出可以表示为如下的公式，其中f为激活函数。

$$o_n^l = f(\sum_m w_{m,n}^l o_m^{l-1} + b_n^l) \tag{13-4}$$

从公式（13-4）中可以看到，不同的神经元对应着不同的线性模型，也就是说权重项和截距项是不同的。但在卷积神经网络里，处理方法是完全不同的，以图 13-10 为例，隐藏层 1 里的神经元都使用相同权重项和截距项，也就是说输入层到隐藏层 1 的连接只使用了 26 个模型参数：25 个权重项以及 1 个截距项。这样的处理在学术上被称为 shared weights and biases（不妨将其翻译为共享参数）。从直观上来讲，这种建模方式表示隐藏层 1 里的所有神经元都在检测同一个图形特征，只是不同神经元检测的位置不一样而已。因此在学术上，常将如图 13-10 所示的结构称为一个 feature map（不妨将其翻译为特征映射）。从模型处理上来讲，这样的方法极大地减少了模型参数的个数[10]，降低了神经网络训练的难度，从而可以搭建和使用层数更多的深度神经网络。

将上面的两种方法结合起来就构成了所谓的卷积层（convolutional layer），根据上面的讨论，卷积层能有效地处理图片中像素点的位置信息。由于一个 feature map 只能处理一种图像特征，为了增强模型识别图像的能力，卷积层通常包含多个 feature map。如图 13-11 所示，卷积层 1 里就有 3 个 feature map，对应着$3 \times 24 \times 24$个神经元，它们一共使用了 3×26 个模型参数。

[10] 在图 13-10 的神经网络中，输入层有 784 个神经元，隐藏层 1 有 576 个神经元。如果使用全连接的神经网络，则输入层到隐藏层 1 就使用了$576 \times (784 + 1) = 452\,160$个模型参数；如果使用 local receptive fields 但不使用 shared weights and biases，则需要$24 \times 24 \times 26 = 14\,976$个模型参数。

书籍是人类进步的阶梯

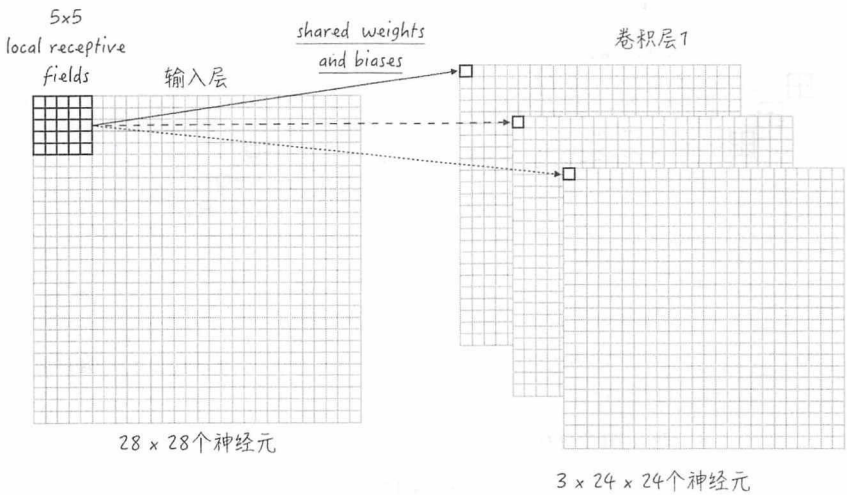


图 13-11

13.2.2 模型结构之池化层

讨论完像素点的位置信息，现在来看看如何在神经网络里模拟人眼对图像的模糊处理。不妨假设 A、B、C、D 是相邻 4 个像素点，如果用一个值（比如这 4 个点的最大值）去代表这 4 个点，那么就相当于将 4 个点模糊地压缩成了一个点，这与 10.1.4 节中所讨论的向量量化（vector quantization）是一样的处理思路。

将这个思路应用到卷积神经网络中，就得到了所谓的池化层（pooling layer）。与神经网络里的其他层不同，池化层里的“神经元”并没有线性模型和激活函数，通常只是简单地求输入信号的最大值（被称为 max-pooling）[11]。

如图 13-12 所示，池化层位于卷积层的后面，它的输入是 feature map 里的神经元。正如上面讨论的，卷积层里通常有多个 feature map，池化层会独立地对每个 feature map 做处理。比如图 13-12 中卷积层里有 3 个 feature map，相应的池化层里就有 3 层 max-pooling。

对于一个 feature map，它里面的神经元将按2×2的正方形区域（pooling 大小）依次与池化层里的神经元相连。但与 local receptive fields 不同的是，这里的正方形区域在移动时相互并不重叠，因此，图 13-12 中一个24×24的 feature map 对应着12×12的 max-pooling。

[11] 除了正文中提到的 max-pooling，池化层还有很多其他的处理方法，比如比较常用的L_2 pooling。

书籍是人类进步的阶梯

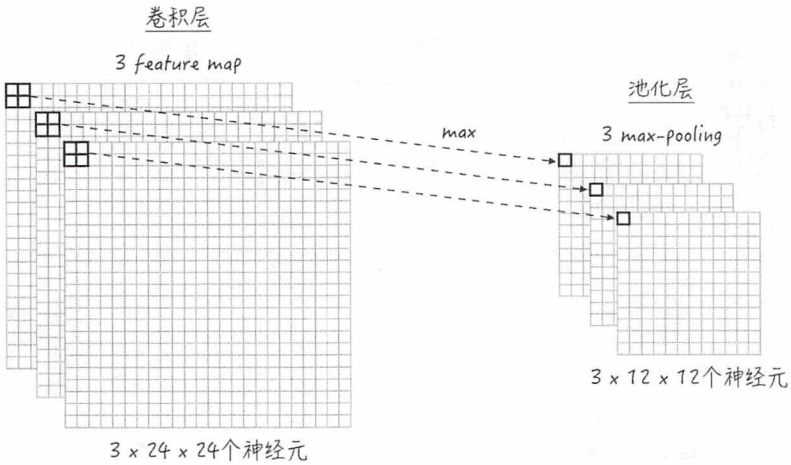


图 13-12

在深度学习里，为了提升模型效果，通常会使用多个卷积层和池化层（两种是成对出现的）。以图 13-13 为例，这个神经网络就有两个卷积层和两个池化层。

在图 13-13 中，卷积层 2 的输入是池化层 1，其中有 4 层 max-pooling。卷积层 2 将把这 4 层 max-pooling 当成重叠着的图片一起做卷积，比如对于卷积层 2 里的每个 feature map，它位于右下角的神经元将与每层 max-pooling 里右下角的 25 个神经元都相连，也就是和 $4 \times 25 = 100$个神经元相连（具体的连接方式可参考如图 13-14 所示的例子）。在实践中，常将叠在一起做卷积的“层”称为卷积层的 input channel，比如对于图 13-13 中的卷积层 2，它有 4 个 input channel，相应地，卷积层 1 有 1 个 input channel；也常将卷积层里的 feature map 称为卷积层的 output channel，比如卷积层 2 就有 12 个 output channel，而卷积层 1 有 4 个 output channel。

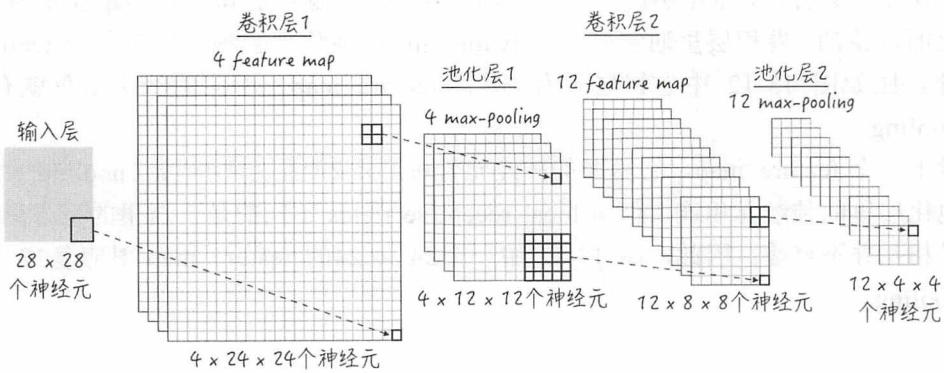


图 13-13

事实上，除了池化层，输入层也可能有好几个渠道（channel），比如对于彩色图片（RGB图片），它相应的输入层里就有 3 个渠道，分别对应着红、绿、蓝 3 种颜色的程度。对于这种情况，卷积层的处理过程是类似的。

为了更直观地理解卷积层在池化层（或输入层）上的作用，可以考虑如图 13-14 所示的例子。图中卷积层有两个 input channel，而 local receptive fields 为2×2的正方形区域。那么卷积层右下角的神经元 A 将与$2 \times (2 \times 2)$个神经元相连（图中用黑色标记的小方块）。不妨假设神经元 A 相应的权重项为$1, -2, 2, 0$和$-3, 4, 5, 3$，截距项为 0，而池化层里相应神经元的输出分别为$1, 2, 0, 2$以及$1, 3, 2, 4$。那么神经元 A 线性部分的输出就等于 28[12]。

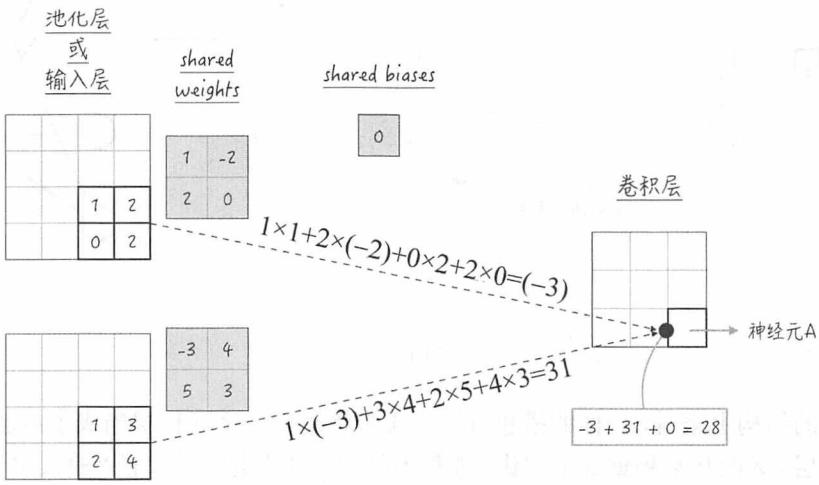


图 13-14

13.2.3 模型结构之完整结构

虽然如图 13-13 所示的神经网络巧妙地模拟了人眼的两个特性，但它还不能被用于图像识别，因为它还不完整，只是卷积神经网络的一部分，下面将讨论后者的完整结构。

跟之前讨论的全连接神经网络类似，需要在网络里定义输出层，以便对标签变量作出预测。通常的做法是在最后一个池化层的后面加上输出层，而且加入的输出层与最后一个池化层是全连接的。事实上，还可以将输出层换成一个全连接的神经网络（输出层可以被看作是层数等于 1 的全连接神经网络），因为实验结果表明，这样的模型效果更好。在学术上，将最后面的全连接神经网络称为全连接层（full-connected layer）[13]。

---

[12] 具体的实现可参考随书配套的代码/ch13-deep_learning/conv2d_example.py。

[13] 在某些文献中，全连接层也被称为 dense layer。

综上所述，完整的卷积神经网络包括 3 个部分：卷积层、池化层和全连接层，具体的结构如图 13-15 所示。值得注意的是，图中池化层 1 与后面的全连接层 1 是全连接的，虽然图中并没有直接地将这些连接表示出来。

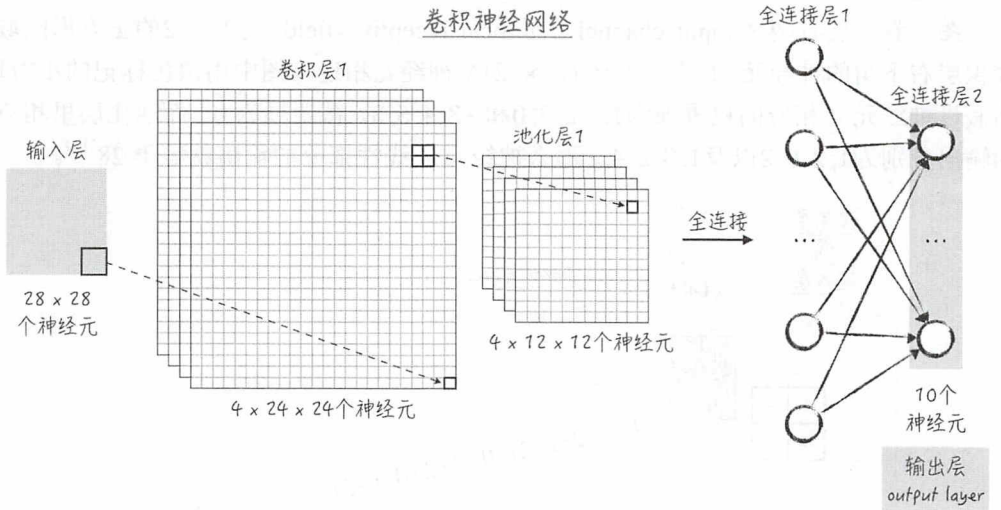


图 13-15

从模型的结构上来看，这样的搭建方法其实是将最后一个池化层当成了后面全连接神经网络的输入层。因此从某种意义上来讲，卷积神经网络主要是对模型的“输入层”做了改进，卷积层和池化层的设计目的主要是为了能从图片里提取出更有效的特征。

13.2.4 代码实现[14]

本节将讨论如何实现卷积神经网络。从代码的角度来讲，卷积神经网络的实现框架与之前讨论的全连接神经网络是类似的，主要的差别在于模型结构的定义，这也是下面讨论的重点。

首先定义卷积神经网络的整体结构，如程序清单 13-5 所示。

（1）由于卷积神经网络将使用像素点的位置信息，因此需要首先将输入数据由 784 维的行向量转换成28 × 28的方阵，如第 10 行代码所示。转换后的数据形状是“[−1, 28, 28, 1]”，其中第一个值等于−1 表示它的第一个维度对应着训练图片的个数，是一个不确定的值，这与 NumPy 中的 array 是类似的；第二个值和第三个值分别表示图片的长和宽；第四个

[14] 完整的实现请参考随书配套的代码/ch13-deep_learning/cnn.py。

书籍是人类进步的阶梯

值表示图片的色系个数，由于图片是黑白的，因此这个值等于 1。在实践中，第四个值常被称为 channel。

（2）卷积层和池化层因为是成对出现的，因此放在一起定义，如第 13～18 行代码所示。这段代码定义的神经网络如图 13-16 所示。这里先抛开实现的具体细节，重点讨论一下参数“filterShape”和“poolSize”。

“filterShape”定义的是卷积层的结构，它是一个四维数组。以卷积层 1 为例（第 13、14 行代码），相应的定义为“filterShape=[5, 5, 1, 20]”。其中前两个值“5, 5”分别表示 local receptive fields 的长和宽；第三个值“1”表示卷积层的 input channel 个数，也就是图片的色系数；第四个值“20”表示卷积层的 output channel 个数，也就是 feature map 的个数。根据这样的参数，卷积层 1 的输出形状为“[−1, 24, 24, 20]”。

“poolSize”定义的是池化层的结构，它也是一个四维数组。以池化层 1 为例（第 13、14 行代码），相应的定义为：“poolSize=[1, 2, 2, 1]”。这个值定义了池化层里窗口（max-pooling）的大小为2 × 2[15]。根据这样的参数，池化层 1 的输出形状为“[−1, 12, 12, 20]”。

类似地可以定义卷积层 2 和池化层 2。值得注意的是，由于池化层 1 的 output channel 等于 20，所以卷积层 2 的 input channel 也等于 20，如第 17、18 行代码所示。

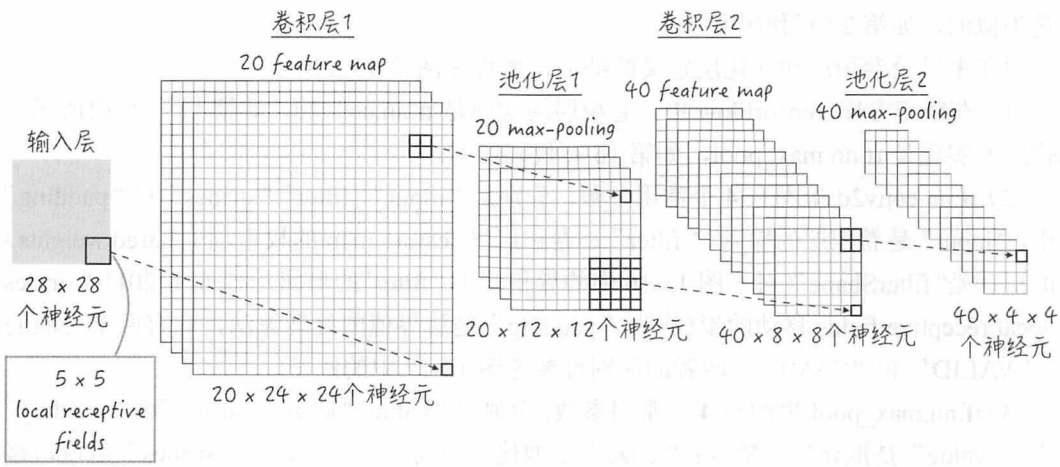


图 13-16

程序清单 13-5 卷积神经网络的结构

```
1 | class CNN(object):
2 |     # 省略掉其他部分
```

[15] 事实上，poolSize 每个值的含义与 img.shape 是类似的：第 1 个值表示样本数、第 2 个值表示窗口的长、第 3 个值表示窗口的宽、第 4 个值表示 channel。因此，一般只改变其中的第 2 个和第 3 个值。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯


```
 3 |
 4 |     def defineCNN(self):
 5 |         """
 6 |         定义卷积神经网络的结构
 7 |         """
 8 |         # MNIST 图片集是一个 784 的行向量，将其转换为 28 × 28 的方阵
 9 |         # 由于图片是黑白的，所以 channel 等于 1，如果是 RGB 彩色图片，channel 应等于 3
10 |         img = tf.reshape(self.input, [-1, 28, 28, 1])
11 |         # 定义卷积层 1 和池化层 1，其中卷积层 1 里有 20 个 feature map
12 |         # convPool1 的形状为[-1, 12, 12, 20]
13 |         convPool1 = self.defineConvPool(img, filterShape=[5, 5, 1, 20],
14 |             poolSize=[1, 2, 2, 1])
15 |         # 定义卷积层 2 和池化层 2，其中卷积层 2 里有 40 个 feature map
16 |         # convPool2 的形状为[-1, 4, 4, 40]
17 |         convPool2 = self.defineConvPool(convPool1, filterShape=[5, 5, 20, 40],
18 |             poolSize=[1, 2, 2, 1])
19 |         # 将池化层 2 的输出变成行向量，后者将作为全连接层的输入
20 |         convPool2 = tf.reshape(convPool2, [-1, 40 * 4 * 4])
21 |         # 定义全连接层
22 |         self.out = self.defineFullConnected(convPool2, size=[30, 10])
```

（3）在卷积神经网络的最后面是它的连接层，由于最后一个池化层需要与全连接层相连，因此首先需要将前者“铺平”，如第 20 行代码所示。全连接层的定义与之前的全连接神经网络是类似的，如第 22 行代码所示。

现在来讨论卷积层和池化层定义的细节，如程序清单 13-6 所示。

（1）在第三方库 TensorFlow 中，卷积层的实现是 tf.nn.conv2d，如第 13 行代码所示；而池化层的实现是 tf.nn.max_pool，如第 16 行代码所示。

（2）tf.nn.conv2d 里面有 4 个常用参数，分别是“input”“filter”“strides”和“padding”。其中，“input”是卷积层的输入；“filter”是卷积层里 feature map 的权重项（shared weights），它的形状是“filterShape”（对于图 13-16 中的卷积层 1，“filter”的形状是[5, 5, 1, 20]）；“strides”是 local receptive fields 移动的步伐[16]；“padding”是边界值的处理算法，它有两个可能的取值：“VALID”和“SAME”。两者的区别可参考图 13-17 中的例子。

（3）tf.nn.max_pool 里面有 4 个常用参数，分别是“value”“ksize”“strides”和“padding”。其中，“value”是池化层的输入；“ksize”是池化层中窗口的大小，而“strides”是窗口移动的步伐，由于池化层中，窗口在移动时并不重叠，因此这两个参数的值常常是相等的；“padding”是边界集的处理算法，具体含义请参考图 13-17。

[16] strides 每个值的含义与 img.shape 是类似的，因此一般只改变其中的第 2 个和第 3 个值。

仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

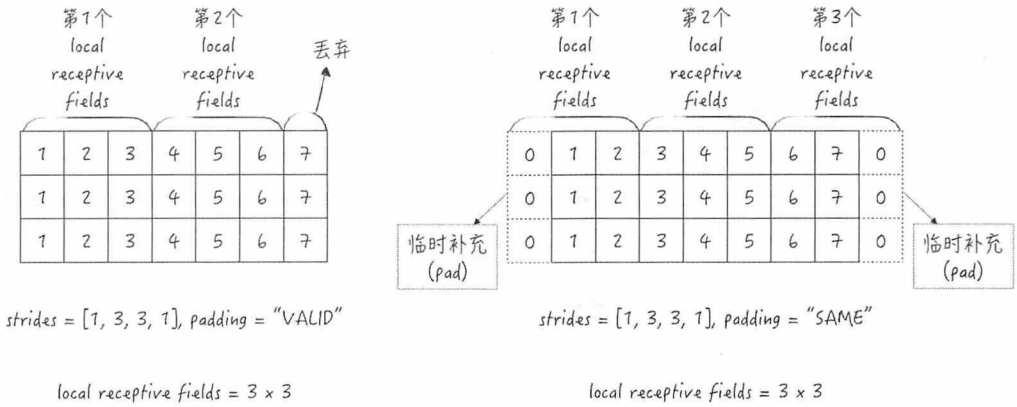


图 13-17

程序清单 13-6 卷积层和池化层

```
 1 | class CNN(object):
 2 |     # 省略掉其他部分
 3 |
 4 |     def defineConvPool(self, inputLayer, filterShape, poolSize):
 5 |         """
 6 |         定义卷积层和池化层
 7 |         """
 8 |         weights = tf.Variable(tf.truncated_normal(filterShape, stddev=0.1))
 9 |         # 将模型中的权重项记录下来，用于之后的惩罚项
10 |         self.W.append(weights)
11 |         biases = tf.Variable(tf.zeros(filterShape[-1]))
12 |         # 定义卷积层
13 |         _conv2d = tf.nn.conv2d(inputLayer, weights, strides=[1, 1, 1, 1],
                        padding="VALID")
14 |         convOut = tf.nn.relu(_conv2d + biases)
15 |         # 定义池化层
16 |         poolOut = tf.nn.max_pool(convOut, ksize=poolSize, strides=poolSize,
padding="VALID")
17 |         return poolOut
```

最后是定义卷积神经网络的全连接层，如程序清单 13-7 所示。全连接层的实现与全连接神经网络并无太大差异。这里只想强调一点：为了防止过拟合，13.1.3 节讨论了 dropout 方法。卷积神经网络同样可以使用这个方法，但通常只在全连接层使用它，如第 26 行代码所示。

程序清单 13-7 全连接层

```
 1 | class CNN(object):
 2 |     # 省略掉其他部分
 3 |
 4 |     def defineFullConnected(self, inputLayer, size):
 5 |         """
 6 |         定义全连接层的结构
```

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯


```
        """
        prevSize = inputLayer.shape[1].value
        prevOut = inputLayer
        layer = 1
        # 定义隐藏层
        for currentSize in size[:-1]:
            weights = tf.Variable(tf.truncated_normal(
                [prevSize, currentSize], stddev=1.0 / np.sqrt(float(prevSize))),
                name="fc%s_weights" % layer)
            # 将模型中的权重项记录下来，用于之后的惩罚项
            self.W.append(weights)
            # 记录隐藏层的模型参数
            tf.summary.histogram("hidden%s" % layer, weights)
            biases = tf.Variable(tf.zeros([currentSize]),
                name="fc%s_biases" % layer)
            layer += 1
            # 定义这一层神经元的输出
            neuralOut = tf.nn.relu(tf.matmul(prevOut, weights) + biases)
            # 对隐藏层里的神经元使用 dropout
            prevOut = tf.nn.dropout(neuralOut, self.keepProb)
            prevSize = currentSize
        # 定义输出层
        weights = tf.Variable(tf.truncated_normal(
            [prevSize, size[-1]], stddev=1.0 / np.sqrt(float(prevSize))),
            name="output_weights")
        # 将模型中的权重项记录下来，用于之后的惩罚项
        self.W.append(weights)
        biases = tf.Variable(tf.zeros([size[-1]]), name="output_biases")
        out = tf.matmul(prevOut, weights) + biases
        return out
```

如果使用上面定义的卷积神经网络对 MNIST 图片集做识别，可以得到如图 13-18 所示的结果。可以看到，相比于 13.1 节中的全连接神经网络，卷积神经网络的模型效果更好。事实上，适当地调整卷积神经网络的结构[17]，可使识别准确率达到 0.992 左右。

模型效果

epoch 1	epoch 15	epoch 30
训练集的准确率 0.894	训练集的准确率 0.983	训练集的准确率 0.988
验证集的准确率 0.893	验证集的准确率 0.979	验证集的准确率 0.983
测试集的准确率 0.901	测试集的准确率 0.981	测试集的准确率 0.985
...	...	

图 13-18

[17] 根据 TensorFlow 官网的例子，如果将卷积层 1 的 feature map 个数设为 32、卷积层 2 的 feature map 个数设为 64、全连接层的结构为[1024, 10]，则模型的识别准确率为 0.992。当然这并不是卷积神经网络的极限，有实验证明，卷积神经网络的准确率可以达到 0.9977。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

13.2.5 结构真的那么重要吗

在上面的章节中，我们动手搭建了全连通神经网络以及卷积神经网络，并使用这两个模型对 MNIST 数字图片集做识别。从书中的例子上来看，卷积神经网络的预测效果更好，但这个结论可信吗？

根据 12.4.4 节中的讨论，深度的全连接神经网络是很难训练的，常常需要几周甚至几个月的时间，因此在实际中，很少有人来耐心等待整个训练过程的完成，但在学术界有这样的人。2010 年的一篇论文[18]搭建了最“原始”的全连接神经网络：网络里的激活函数是 sigmoid 函数，整个网络的结构为[2500, 2000, 1500, 1000, 500, 10]。如果使用这个神经网络对 MNIST 图片集进行识别，预测的准确率可以达到惊人的 0.9965，也就是说模型只预测错了 35 张图片（测试集里有 10 000 张图片），这比很多卷积神经网络的预测效果都好。模型预测错误的 35 张图片如图 13-19 所示，可以看到，即使是人来识别这些数字图片也很容易犯错。

图 13-19

按照人类“想当然”的理解，卷积神经网络是更接近人眼的模型，因此它的识别效果更好。但上面的结果却暗示识别效果的提升并不是来源于模型结构的创新，而是更多地来自计算机计算能力的提高以及更加高效的近似算法。

---

[18] Ciresan D C, Meier U, Gambardella L M, et al. Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition[J]. Corr, 2010, 22(12):3207 - 3220.

为了提高模型效果，这篇论文使用了 deforming images 的手段来扩充训练集，这些技术手段包括旋转图片、平移图片等。当然这些技术并没有什么独特性，是图片识别领域常用的方法。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

从理论的角度来讲，上面的结果表明，到目前为止，人类并没有理解哪怕最简单的神经网络，我们不知道这类模型的原理是什么、它的适用范围有多大，以及它的极限在哪里。但从实用的角度来讲，预测效果的改进已经足够吸引人，业界也在积极尝试将神经网络应用到实际的生产中。在我看来，这样的局面是有些令人担忧的，因为这有可能造成意想不到的后果。当然如果能从理论上彻底理解神经网络，那么它在实际中的应用前景将不可估量。

13.3 其他深度学习模型

截止到这里，我们仔细讨论了在图片识别领域应用很广的卷积神经网络，但它只是深度学习浩瀚繁星中的一颗，还有很多有趣又有效的深度学习模型。下面将简单介绍这其中的两类：一类是擅长处理时序数据的递归神经网络；一类是能分析没有标签变量的非监督式神经网络。与之前的章节不同，下面只介绍模型的整体结构，具体的细节则不做展开。

13.3.1 递归神经网络

之前讨论的全连接神经网络和卷积神经网络虽然差异很大，但它们有一个共同点：模型的网络里并无有向环。换句话说，模型里一个神经元发出的信号只能向前传播，信号无法经过几次传递后又再次回到发出它的那个神经元。在学术上，这样的神经网络被称为前馈神经网络（feedforward neural network）。

前馈神经网络的模型结构决定了它并不能很好地处理数据之间的前后相关性。以上面讨论的图片识别为例，将训练集里的图片顺序打乱并不会影响模型识别图片的能力，也就是说，模型并没有考虑图片之间的顺序。当然在图片识别的例子中，图片的顺序并不能提供什么额外信息，但在一些其他的应用场景里，数据之前的前后次序却是至关重要的。比如在人类的语言里，每个文字的意义在很大程度上取决于它的上下文，而它本身又是其他文字的上下文；又比如在股票市场中，最近一段时间的股票价格走势会在很大程度上影响股票未来的表现。

对于这样的应用场景，前馈神经网络就不再适用了，我们需要搭建动态的神经网络。也就是说，在这个新搭建的动态神经网络中，当前神经元的输出不但能影响后面神经元的状态，还要能通过网络影响它前面的神经元。这样就相当于在神经网络里建立起了时间或者上下文的概念，因为当前神经元的当前状态其实与它之前的状态相关。这样的动态神经网络在学术上被称为递归神经网络（Recurrent Neural Network，RNN）。

递归神经网络是数据科学领域的研究热点，它包含很多形状各异的模型，但遗憾的是，这些模型大多难以训练，难以使用。在实际生产中应用较广的是一类特殊的递归神经网络——长短期记忆（Long Short-Term Memory，LSTM）。它被广泛用于语音识别、自然语言处理和机

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

器翻译等领域。

13.3.2 长短期记忆

与前馈神经网络的讨论类似，首先来看看如何用图像来表示长短期记忆。如图 13-20 所示，这个模型通常被表示为链式的图形，这跟 9.4 节中讨论的隐马尔可夫模型（HMM）比较类似。

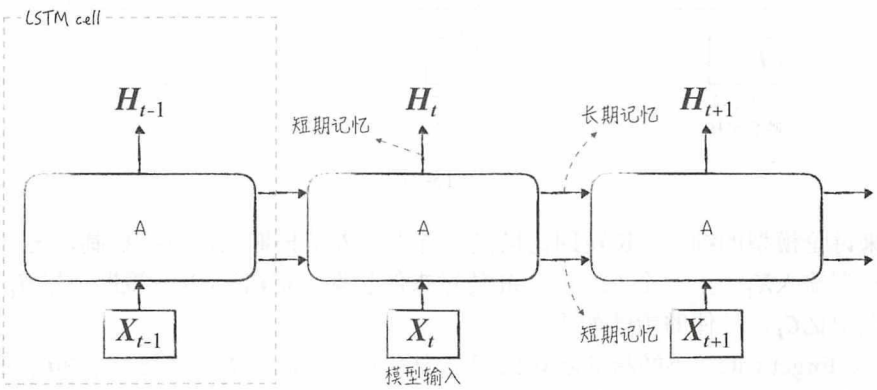


图 13-20

在图 13-20 中，A 表示模型的处理逻辑、$\boldsymbol{X_t}$表示模型在t时刻的输入、$\boldsymbol{H_t}$表示模型在t时刻的输出[19]，这 3 者加在一起构成了所谓的 LSTM cell（不妨将其翻译为“记忆单元”）。不同的 LSTM cell 之间传递两种信息（在数学上表示为行向量），一是所谓的长期记忆，另一个是所谓的短期记忆。顾名思义，长期记忆关注的是从开始到当前的所有信息，而短期记忆更关注当前的信息。在学术文献里，长期记忆被称为 cell state，记为$\boldsymbol{C_t}$，而短期记忆被称为 hidden state，记为$\boldsymbol{H_t}$，在之后的讨论里将沿用这样的记号[20]。

需要注意的是，虽然每个 LSTM cell 都有相应的模型输出，但在实际应用中，我们可能只关心其中一个或几个输出。以股票市场为例，我们希望能根据前面 3 个交易日的股票价格预测当天的股票价格。那么在这种应用场景下，长短期记忆模型里将对应地有 3 个 LSTM cell，但只有最后一个 LSTM cell 的输出才是模型所关心的。换句话说，只有这个预测值才会被计入模型的损失函数，如图 13-21 所示。

---

[19] 之所以用$\boldsymbol{H_t}$来表示模型的输出，是因为 LSTM cell 可以类比为隐藏层，它的输出需要经过数学变换（比如 softmax 函数）才能得到最终的模型预测。

[20] 在有些文献中，长期记忆被称为 long-term memory，短期记忆被称为 working memory。

书籍是人类进步的阶梯 仅供非商业用途或交流学习使用

书籍是人类进步的阶梯

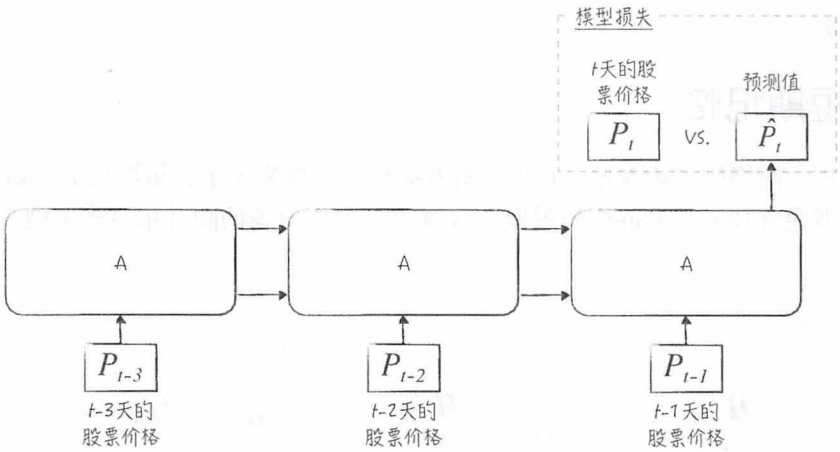


图 13-21

现在来讨论模型的细节。长短期记忆模型首先引入了长期记忆更新机制，这个机制将根据当前的模型输入$\boldsymbol{X_t}$、上一个 LSTM cell 传过来的长期记忆$\boldsymbol{C_{t-1}}$以及短期记忆$\boldsymbol{H_{t-1}}$，得到最新的长期记忆$\boldsymbol{C_t}$，具体的步骤如下。

- 定义 forget gate（不妨将其翻译为记忆力度）[21]，记为$\boldsymbol{R_t}$，它是一个和$\boldsymbol{C_t}$同样大小的行向量，里面的元素是 0～1 的数。元素值等于 1 表示完全保留之前的长期记忆；这个值等于 0 则表示完全丢弃之前的长期记忆。具体的公式如下所示，公式中的f为 sigmoid 函数。

$$\boldsymbol{R_t} = f(\boldsymbol{W_r} \circ \boldsymbol{X_t} + \boldsymbol{U_r} \circ \boldsymbol{H_{t-1}} + \boldsymbol{b_r}) \tag{13-5}$$

- 定义 input gate（不妨将其翻译为更新力度），记为$\boldsymbol{I_t}$，以及候选的新增记忆$\boldsymbol{NC_t}$。其中，$\boldsymbol{I_t}$也是一个和$\boldsymbol{C_t}$同样大小的行向量，其元素值是 0～1 的数字，它表示多大比例的新增记忆将被纳入长期记忆。具体的公式如下：

$$\begin{aligned} \boldsymbol{I_t} &= f(\boldsymbol{W_i} \circ \boldsymbol{X_t} + \boldsymbol{U_i} \circ \boldsymbol{H_{t-1}} + \boldsymbol{b_i}) \\ \boldsymbol{NC_t} &= \tanh(\boldsymbol{W_n} \circ \boldsymbol{X_t} + \boldsymbol{U_n} \circ \boldsymbol{H_{t-1}} + \boldsymbol{b_n}) \end{aligned} \tag{13-6}$$

- 定义长期记忆的更新机制，将上面两点结合可以得到如下的更新公式：

$$\boldsymbol{C_t} = \boldsymbol{R_t} \circ \boldsymbol{C_{t-1}} + \boldsymbol{I_t} \circ \boldsymbol{NC_t} \tag{13-7}$$

在长期记忆更新机制之后是短期记忆的产生机制，这个机制将根据最新的长期记忆$\boldsymbol{C_t}$、当前模型输入$\boldsymbol{X_t}$以及之前的短期记忆$\boldsymbol{H_{t-1}}$，得到最新的短期记忆$\boldsymbol{H_t}$，具体的步骤如下。

- 定义 output gate（不妨将其翻译为输出力度），记为$\boldsymbol{O_t}$，它与$\boldsymbol{I_t}$非常类似，可以近似地理解为，它表示有多大比例的长期记忆将转换为短期记忆，具体的公式如下：

$$\begin{aligned} \boldsymbol{O_t} &= f(\boldsymbol{W_o} \circ \boldsymbol{X_t} + \boldsymbol{U_o} \circ \boldsymbol{h_{t-1}} + \boldsymbol{b_o}) \\ \boldsymbol{H_t} &= \boldsymbol{O_t} \circ \tanh \boldsymbol{C_t} \end{aligned} \tag{13-8}$$

[21] 虽然 forget gate 名字里面有 forget 这个单词，但它表示的却是对长期记忆的保留程度。

书籍是人类进步的阶梯

书籍是人类进步的阶梯

将上面的步骤总结一下，可以得到如图 13-22 所示的模型处理流程图。

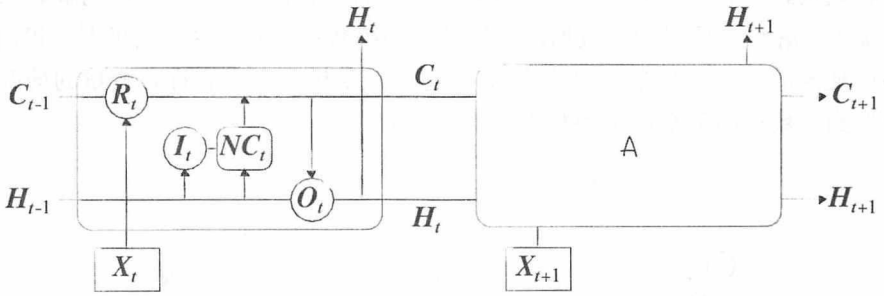


图 13-22

第三方库 TensorFlow 实现了这个模型，具体的类是 tf.contrib.rnn.BasicLSTMCell。限于篇幅，代码的细节在这里就不做讨论了。只想提醒读者注意的是在 TensorFlow 的实现中，$\boldsymbol{H_t}$和$\boldsymbol{C_t}$的大小（行向量的长度）通常是相同的，用参数 num_units 表示。

13.3.3 非监督式学习

前馈神经网络和递归神经网络都是监督式学习，它们非常依赖数据里的标签变量。如果数据里标签变量质量不好，或者标签变量缺失，则模型的效果会大打折扣，当然如果完全没有标签变量，那么这些模型都将“巧妇难为无米之炊”了。但是在现实生活中，标签数据是很稀缺的，而且往往质量难以保证，这极大地限制了深度学习的应用和发展。因此，学术界一直在想办法搭建适应非监督式学习的深度神经网络。

与传统的神经网络相比，非监督式神经网络由于没有标签变量，因此并没有直观的方法来评估模型结果的好坏。换句话说，非监督式神经网络最大的难点在于定义模型的损失函数。当然这并不是我们第一次遇到这样的难题了，在 9.4 节讨论隐马尔可夫模型时就曾碰到类似的问题。当时的解决方案是按照生成式模型（generative model）框架，定义模型的似然函数（likelihood function），然后使用最大似然估计法（Maximum Likelihood Estimation，MLE）来求解模型的参数。

这个方案其实是通用的，可以借助它搭建非监督式神经网络，其中最典型的代表就是变分自编码（Variational Autoencoder，VAE），它的模型结构如图 13-23 所示。从模型类别上来看，VAE 属于全连接神经网络。它的网络结构通常是对称的，输入层和输出层的神经元个数是相同的，模型训练的目标是让输入层的输入（训练数据）与输出层的输出越相似越好。

位于网络最中间的隐藏层对应着数据的隐藏状态（通常这一层的神经元也是最少的），这和隐马尔可夫模型中的隐藏状态层是类似的。可以这样理解模型的运作机制，训练数据首

书籍是人类进步的阶梯

先通过神经网络的前半部分，找到相应的隐藏状态，这一步被称为编码（encode）；然后根据得到的隐藏状态和后半部分神经网络得到还原之后的数据，这一步被称为解码（decode）。理想的情况下，解码之后得到的数据近似地等于训练数据。通过这样的模型，我们可以得到数据本不可观测的类别（数据的隐藏状态），也就是说模型在没有标签变量的情况下，自动地“领悟”到了数据的类别，仿佛和人一样。

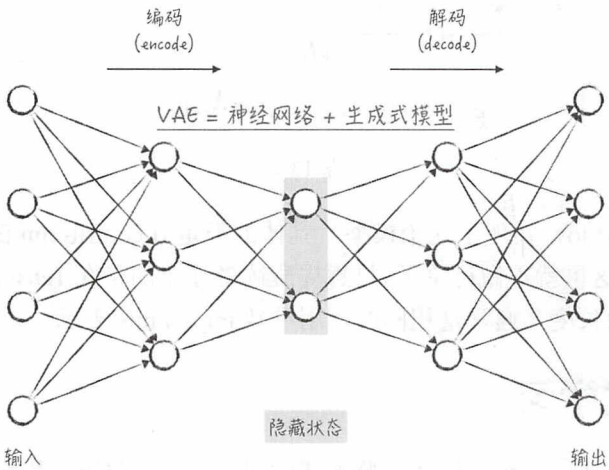


图 13-23

除了 VAE 之外，非监督式神经网络中还有一个非常有意思的模型——生成对抗网络（Generative Adversarial Network，GAN）。这个模型可以将图片转换为某位艺术家的风格，以图 13-24 为例，模型将图片转换为了梵高的风格，仿佛它能画画一样。当然，非监督式神经网络还有很多其他模型，限于篇幅，这里就不一一讨论了。

图 13-24[22]

[22] Gatys L A, Ecker A S, Bethge M. A Neural Algorithm of Artistic Style[J]. Computer Science, 2015.

13.4 本章小结

本章是全书的最后一章，讨论的内容是深度学习，也就是层数较多的深度神经网络。

由于神经网络是非常复杂的模型，所以极易引起过拟合的问题。本章首先以数字图片识别为例，讨论了防止神经网络发生过拟合的常用技术手段。与其他模型类似，防止过拟合最直接的方法就是在模型的损失函数里加入惩罚项，常用的有 L1 和 L2 惩罚项。但对于特别大型的神经网络，惩罚项还不足以解决过拟合问题，需要使用特殊的 dropout 方法。这个方法将在训练神经网络的过程中，随机地删掉一些神经元，防止模型过度优化。

经典的全连接神经网络虽然也能用于图像识别，而且模型效果还不错，但从结构上来看，它无法模拟人眼局部识别和模糊处理的能力。为此，本章接着讨论了卷积神经网络。这个模型在全连接神经网络的基础上增加了卷积层和池化层，其中卷积层的设计目的是为了更好地捕捉图片的局部特征，而池化层则是为了模拟人眼的模糊处理能力。由于这两个特殊的设计，模型的预测效果有了进一步的提升，而且从工程实现的角度来讲，卷积层和池化层由于是局部连接，有效地减少了模型参数，从而降低了模型训练的难度，使得我们能在实践中搭建和使用层数更多的深度神经网络。

全连接神经网络和卷积神经网络都属于前馈神经网络，这类模型是静态的，因此无法有效地处理动态数据。为此我们接着介绍了另外一类神经网络——递归神经网络。并重点讨论了其中的长短期记忆模型，后者在语音识别和时间序列分析领域取得了较好的预测效果。

本章最后讨论了神经网络在非监督式学习里的应用。由于没有标签变量，非监督式神经网络通常需要依赖生成式模型框架定义模型的似然函数，并使用最大似然估计法进行求解，其中最典型的例子是 VAE 和 GAN。

与数据科学领域的经典模型相比，本章讨论的深度学习模型是这门学科中最前沿的内容（虽然它们看起来并不那么“智能”，而且还有不少明显的缺陷）。这意味着这些模型还不够成熟，在非互联网领域，比如金融、工业等，它们的应用还十分有限。

由于本书并不是一本专门讲解深度学习的书籍，所以本章只讨论了其中最基础，也最经典的内容，想要了解更多的读者可参考其他书籍，比如 Ian Goodfellow 等人编写的 *Deep Learning*。

数据科学是一门与大数据和人工智能相关的新兴学科。它的发展十分迅猛，在越来越多的行业里发挥着重要作用。然而这门学科的学习难度很大，因为它包含的知识点多、跨度大，涉及计算机编程和数据建模等多个方面。而且它的更新速度很快，要求该领域的人员不断学习新知识，掌握新技能。

为了更有效地学习，优秀的学习资料必不可少。虽然网上有不少相关的技术文章，各开源算法库的使用示例也很多，但这些资料都比较碎片化，不成体系，难以给读者勾勒出关于数据科学的全景图。作者结合实际工作中的经验，用本书呈现了数据科学较全面的知识体系，希望能带给读者不同的阅读体验，并帮助读者更好地在数据中挖掘“金矿”。

数据科学是一门交叉学科，涉及数理统计、代码编程、商业分析等多个领域的知识。我们正在从 IT 时代步入 DT 时代，数据科学将扮演越来越重要的角色，企业对数据科学家的需求也将快速增加。数据科学家被《哈佛商业评论》评为“21 世纪最吸引人的职业”，可见其魅力所在。唐亘的这本书系统介绍了目前数据科学领域的核心知识和技能，帮助读者搭建一个系统的知识体系。我把它推荐给对数据科学感兴趣或者立志成为下一位数据科学家的你！

GrowingIO 创始人兼 CEO，曾任 LinkedIn 美国商业分析部高级总监，
《首席增长官》一书作者　张溪梦（Simon Zhang）

图灵奖获得者 Jim Gray 将数据科学称作科学研究的“第四范式”（the fourth paradigm）。数据科学不仅会影响到科学的各个方面，也会在各领域的应用中发挥重要的作用。唐亘以其坚实的数据科学基础和多年的大数据分析经验，用浅显易懂的方式撰写了这本《精通数据科学：从线性回归到深度学习》。这本书没有局限于坐而论道，让读者对各种模型有恐惧感，而是通过应用实例将问题、概念、模型和解决方案有机地联系起来，使读者能够快速理解和应用数据科学。对于数据科学的学习者和不同领域的应用者来说，这本书非常值得一读。

复旦大学教授，博士生导师，
复旦大学航空航天数据研究中心主任　杨卫东

将一本技术书籍写得通俗易懂而又深刻透彻是很难的，但唐亘这本《精通数据科学：从线性回归到深度学习》做到了这一点。本书从技术、方法、实践这 3 个维度系统地介绍了数据科学的方方面面，内容详实，解读清晰，细节与全貌兼顾，既适合初学者阅读，也可以作为深入研究的参考用书。

美国罗格斯大学管理科学及信息系统系终身教授，
中国计算机学会大数据专家委员会委员　林晓东

异步社区 www.epubit.com
异步社区 www.epubit.com
新浪微博 @人邮异步社区
投稿/反馈邮箱 contact@epubit.com.cn

ISBN 978-7-115-47910-5
9 787115 479105 >
ISBN 978-7-115-47910-5
定价：99.00 元

封面设计：广领设计
分类建议：计算机／数据科学
人民邮电出版社网址：www.ptpress.com.cn